Safe curves for elliptic-curve cryptography

Daniel J. Bernstein[1,2] and Tanja Lange[3,2]

[1] University of Illinois at Chicago, USA
[2] Academia Sinica, Taiwan
djb@cr.yp.to
[3] Eindhoven University of Technology, Netherlands
tanja@hyperelliptic.org

Abstract. This paper surveys interactions between choices of elliptic curves and the security of elliptic-curve cryptography. Attacks considered include not just discrete-logarithm computations but also attacks exploiting common implementation pitfalls.

1 Introduction

In the original 1976 Diffie–Hellman (DH) key-exchange system [91, Section 3], user 1 has secret key X_1 and public key $Y_1 = g^{X_1}$; user 2 has secret key X_2 and public key $Y_2 = g^{X_2}$; and so on for any number of users. Here g is a "fixed primitive element of $GF(q)$", i.e., a generator of the multiplicative group $\mathbb{F}_q^*$, where q is a fixed prime number. To communicate with user i, user j computes a shared secret $g^{X_i X_j}$ as $Y_i^{X_j}$, and uses this shared secret as a key for symmetric-key cryptography to encrypt and authenticate messages. User i computes the same shared secret as $Y_j^{X_i}$.

The DH system is broken if an attacker can solve the "discrete-logarithm problem" for the group $\mathbb{F}_q^*$, the problem of computing X_i from g^{X_i}. The 1976 paper commented that "for certain carefully chosen values of q" this problem "requires on the order of $q^{1/2}$ operations, using the best known algorithm".

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was supported in part by the U.S. National Science Foundation under grant 1018836, the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy–EXC 2092 CASA–390781972 "Cyber Security in the Age of Large-Scale Adversaries", the Academia Sinica Grand Challenge Seed Project AS-GCS-113-M07, the European Commision through the Horizon Europe program under project number 101135475 (TALER), and the Dutch Research Council (NWO) under grants 613.009.144 and 639.073.005. "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation" (or other funding agencies). Permanent ID of this document: 47f17793322fcaf3ef67f982e2c3ae86b34c17ce. Date: 2024.08.09.

The discrete-logarithm algorithms cited in that paper are "generic" algorithms that work for any group. They use about $\ell_1^{1/2}+\ell_2^{1/2}+\cdots$ operations if the order of the generator factors into primes $\ell_1,\ell_2,\ldots$ (not necessarily distinct). A careful choice of q for the DH system then takes, e.g., $q-1=2\ell$ for a prime ℓ, so that the attacks take about $\ell^{1/2}\approx 0.7q^{1/2}$ operations; i.e., the DH security level against these attacks is about $(1/2)\log_2 q$ bits, where security level means logarithm base 2 of attack cost.

However, further analysis showed that a non-generic method called "index calculus"—which had been introduced in 1922 by Kraitchik [**154**, page 120], and which had been used at a much larger scale by Western and J. Miller [**226**]—had much better scalability in solving discrete-logarithm problems, reducing the security level to $(\log_2 q)^{1/2+o(1)}$ bits asymptotically. Index calculus applies more generally to discrete logarithms in $\mathbb{F}_q^*$ for arbitrary prime powers q, not just primes, and applies to integer factorization (see, e.g., Kraitchik's 1926 book [**155**, Chapitre XIV]), in all cases reducing the security level to $(\log_2 q)^{1/2+o(1)}$ bits.

There have been many further developments of index calculus since then. The security level dropped in the early 1980s to $(\log_2 q)^{1/3+o(1)}$ when q is a power of 2, and then a decade later to $(\log_2 q)^{1/3+o(1)}$ when q is a prime, and then two decades after that to $(\log_2 q)^{o(1)}$ when q is a power of 2; see generally [**121**]. Each of these exponent changes was preceded and followed by many speedups that had less effect on the asymptotics but that pushed many specific sizes of q across the line from "unbroken" to "broken".

1.1 Elliptic-curve cryptography (ECC)

The central idea of ECC is to *stop index calculus*, specifically by replacing the groups $\mathbb{F}_q^*$ in the DH system with elliptic-curve groups $E(\mathbb{F}_q)$. This idea was introduced in a 1986 paper by V. Miller [**174**] and, independently, a 1987 paper by Koblitz [**152**]. Miller gave arguments that index calculus "is not likely to work on elliptic curves". Koblitz wrote that "the analog of the discrete logarithm problem on elliptic curves is likely to be harder than the classical discrete logarithm problem, especially over $\mathrm{GF}(2^n)$".

The bird's-eye view is that this has been remarkably successful. Most cryptographic applications today rely on ECC, although there are still some uses of RSA. Computations in $E(\mathbb{F}_q)$ take more effort than computations in $\mathbb{F}_q^*$, but this is outweighed by the fact that for $E(\mathbb{F}_q)$ we can take q just large enough to resist generic discrete-logarithm algorithms, while for $\mathbb{F}_q^*$ we need much larger q to resist index calculus. More importantly, $E(\mathbb{F}_q)$ has maintained its security level much better than $\mathbb{F}_q^*$ has, and thus inspires more confidence than $\mathbb{F}_q^*$ does.

A closer look shows, however, that there have been many successful attacks against specific elliptic-curve cryptosystems—including some curves that had been specifically proposed for speed reasons. For example, "pairings" were shown in the mid-1990s (see Section 5.1) to reduce discrete logarithms in $E(\mathbb{F}_q)$ to discrete logarithms in $\mathbb{F}_q^*$ if $\#E(\mathbb{F}_q)=q-1$; papers a few years later (see also Section 5.1) showed how to reduce discrete logarithms in $E(\mathbb{F}_q)$ to very easy

discrete logarithms in $\mathbb{F}_q$ (not $\mathbb{F}_q^*$) if $\#E(\mathbb{F}_q) = q$; and further classes of "weak curves" are known when q is not a prime (see Section 2.1), where "weak" means that discrete logarithms involve noticeably fewer than $q^{1/2}$ operations.

A wave of standards then appeared specifying how to choose curves for ECC: ANSI X9.62 in 1999 [8], IEEE P1363 in 2000 [134], SEC 2 in 2000 [203], NIST FIPS 186-2 in 2000 [220], and ANSI X9.63 in 2001 [9]. Later elliptic-curve standards include Brainpool in 2005 [66], NSA Suite B in 2005 [184], and ANSSI FRP256V1 in 2011 [86]. These standards generally leave a wide berth around all known classes of weak curves; the goal is to make sure that the elliptic-curve discrete-logarithm problem (ECDLP) is very difficult.

1.2 ECC security risks

There are three important caveats regarding these ECC standards. First, it seems increasingly likely that future attackers will have quantum computers, breaking all of these cryptosystems. Second, it is conceivable that there are further classes of weak curves. Third, there are many attacks that break real-world ECC without solving ECDLP.

This third problem is not a future problem; it is not a problem that relies on the *possibility* of better ECDLP attacks; it is a real failure mode that we have seen again and again. None of the standards mentioned above do a good job of ensuring ECC security. See Appendix A for a chronology of ECC vulnerabilities.

We posted a "SafeCurves" web site [43] in 2013 aimed at improving the situation:

- The web site highlights the "core problem", namely that "**if you implement the standard curves, chances are you're doing it wrong**: Your implementation produces incorrect results for some rare curve points. Your implementation leaks secret data when the input isn't a curve point. Your implementation leaks secret data through branch timing. Your implementation leaks secret data through cache timing."
- The web site explains that these problems "are exploitable by real attackers, taking advantage of the gaps between ECDLP and real-world ECC: ECDLP is non-interactive. Real-world ECC handles attacker-controlled input. ECDLP reveals only nP. Real-world ECC also reveals timing (and, in some situations, much more side-channel information). ECDLP always computes nP correctly. Real-world ECC has failure cases. Secure implementations of the standard curves are theoretically possible but very hard."
- The web site explains that better curve choices help: "Most of these attacks would have been ruled out by better choices of curves that allow *simple* implementations to be *secure* implementations. This is the primary motivation for SafeCurves. **The SafeCurves criteria are designed to ensure ECC security, not just ECDLP security.**"

Note the word "most" here: better curve choices make ECC implementation failures less likely and are the focus of SafeCurves, but one needs to take further

steps to eliminate the remaining failures. The web site notes, for example, that some attacks "would have been ruled out by better choices at higher levels of ECC protocols", such as the way that randomness is used in signature systems.

Ten years of further experience with ECC security failures have confirmed the importance of the SafeCurves criteria, in much the same way that decades of improvements in index calculus have confirmed the importance of switching from $\mathbb{F}_q^*$ to $E(\mathbb{F}_q)$. The point is not that everything is broken on one side of the line—it is certainly not true that *all* $\mathbb{F}_q^*$ have been broken, or that *all* implementations of curves failing the SafeCurves criteria have been broken—but rather that the switch reduces security risks.

1.3 This paper

The intention of this paper is to be "the SafeCurves paper": we present and justify the SafeCurves criteria, giving additional explanation and updated examples of why the criteria are important.

The fact that different curve choices have different effects on implementation simplicity is not something obvious: it relies on many years of research into elliptic-curve computations. The paper explains the relevant features of these computations, for example explaining why state-of-the-art ECC software uses formulas introduced in 1987 by Montgomery [**177**] in some situations and uses formulas introduced in 2008 by Hisil, Wong, Carter, and Dawson [**130**] in other situations.

The paper is organized into two parts. The first part is motivated by ECDLP attacks: Sections 2, 3, 4, 5, 6, and 7 cover, respectively, the field of definition, the curve equation, the order of the base point, the embedding degree, the CM field discriminant, and rigidity.

We emphasize that the SafeCurves criteria reject curves that are known to have faster ECDLP attacks, *and* proactively reject many further curves just in case there are further ECDLP attacks beyond what has been discovered. This is common practice in ECC standards, as noted above, but these sections point out some differences in the details.

The second part is motivated by ECC attacks *beyond* ECDLP attacks: specifically, Sections 8, 9, 10, and 11 cover requirements on "ladders", "twists", "completeness", and "indistinguishability". This is the central contribution of SafeCurves.

The general theme of this part is that there are tensions between the security, simplicity, and speed of ECC software—and these tensions are larger for some curves than for others. This matters because implementors are typically writing the simplest code that they can that passes some tests, and then modifying it for speed if there are speed complaints (or fears of speed complaints). Implementors can sometimes be convinced to take extra steps for security, especially if there are tools enforcing those steps; but needing more of those steps means more chances of disaster.

We do not repeat the tables of SafeCurves ratings here; the tables are easier to use on the web site [**43**]. However, for concreteness, we list three representative

name	curve equation over $\mathbb{F}_p$	prime p	source
NIST P-256	$y^2 = x^3 - 3x + b$; "large" b	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	[220]
brainpoolP256t1	$y^2 = x^3 - 3x + b$; "large" b	"dense"	[66]
Curve25519	$y^2 = x^3 + 486662x^2 + x$	$2^{255} - 19$	[23]

Table 1.3.1. Three examples of curves that have been proposed for cryptographic use. The "dense" p and "large" b values are listed in Sections 2 and 3 respectively.

examples of curves in Table 1.3.1. SafeCurves classifies one of these examples, Curve25519, as safe, meaning that the curve meets all of the SafeCurves criteria.

1.4 Context

We do not claim credit for the general idea of investigating the security consequences of implementation pitfalls. This is one of the main topics of the security literature. Regarding ECC in particular, this paper cites many examples of vulnerabilities in ECC implementations.

However, the processes used to select cryptosystems often assume perfect implementations. For example, NSA wrote the following in 1992 (see [7]) regarding the new NSA/NIST DSA proposal (and regarding the existing Data Encryption Standard, which remained a standard until 2005):

> We are unaware of any weaknesses in the DSS or in the DES when properly implemented and used for the purposes for which they both are designed.

Similarly, regarding the NIST curves, NIST wrote the following in 2019 [183]:

> NIST is not aware of any vulnerabilities to attacks on these curves when they are implemented correctly and used as described in NIST standards and guidelines.

Regarding better curves, [183] wrote that "their designers claim that they offer better performance and are easier to implement in a secure manner"; [183] did not cite any of the literature *demonstrating* the performance benefits and ease of secure implementation of these curves, and did not mention the likelihood and consequences of insecure implementation of the NIST curves.

There have been some recommendations to adjust cryptosystem-selection processes to predict and avoid implementation pitfalls. For example, one of Rivest's comments [204] in 1992 regarding DSA was as follows: "The poor user is given enough rope with which to hang himself—something a standard should not do." What we are asking in this paper is how curve choices influence the chance of implementations being secure.

This paper incorporates curve-selection material from documents we have previously posted, including [42], [43], [44], and [46]. Some of those documents look beyond curve choices at how other ECC design choices affect the speed, simplicity, and security of implementations.

2 Fields

To specify an elliptic curve to use in ECC, one specifies a prime number p and then an elliptic-curve equation "over" the finite field $\mathbb{F}_p$, i.e., an elliptic-curve equation with coefficients in $\mathbb{F}_p$.

For example, the NIST P-224 curve is defined over $\mathbb{F}_p$ where $p = 2^{224} - 2^{96} + 1$; the NIST P-256 curve is defined over $\mathbb{F}_p$ where $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$; Curve25519 is defined over $\mathbb{F}_p$ where $p = 2^{255} - 19$; and brainpoolP256t1 is defined over $\mathbb{F}_p$ where

$$p = 76884956397045344220809746629001649093037950200943055203735601445031516197751.$$

See Table 1.3.1 and Section 3 for the elliptic-curve equations, and see the SafeCurves web site for further examples.

There are other types of elliptic curves. In particular, there are many ECC papers that consider elliptic curves over non-prime finite fields. However, the SafeCurves criteria require prime fields.

ECC standards from the turn of the century typically allowed binary curves. For example, NIST's FIPS 186-2 digital-signature standard specified 15 curves, of which 10 were defined over binary fields. However, the latest version of that standard, FIPS 186-5 from 2023, says that "Elliptic curves defined over binary curves ... are now deprecated". The Brainpool standard and NSA's Suite B standards had already required prime fields.

2.1 Is ECDLP broken for non-prime fields?

The security story for ECDLP over non-prime fields (e.g., binary extension fields $\mathbb{F}_q$ with $q = 2^n$) is more complicated and less stable than the security story for ECDLP over prime fields, as illustrated by [104], [116], [114], [89], and [195].

These attacks construct various non-prime fields $\mathbb{F}_q$ and elliptic curves over $\mathbb{F}_q$ for which attacks exploiting proper subfields of $\mathbb{F}_q$ break ECDLP noticeably faster than generic attacks do. There are some choices of q for which all curves are weak in this sense, and there are even some choices of q for which all curves are broken in time subexponential in $\log q$. On the other hand, there are still many ways to choose non-prime fields and curves that avoid these attacks.

We reiterate that security requirements go beyond avoiding known attacks. Requiring prime fields is *not* making the incorrect claim that non-prime fields are categorically broken; it is instead simplifying attack analysis by excluding the subfield structure used in various attacks. This removes cases where the known attacks apply, *and* proactively removes further cases that share the same structure.

One of the authors of this paper commented in 2006, in the paper [23] introducing Curve25519, that prime fields "have the virtue of minimizing the number of security concerns for elliptic-curve cryptography". There is general agreement that prime fields are the safe, conservative choice for ECC.

2.2 Are "special" primes dangerous?

Index calculus for $\mathbb{F}_p^*$ relies on the fact that integers often factor into small primes. The chance of this happening depends on how large the integers are. Optimized index calculus for $\mathbb{F}_p^*$ using the "number-field sieve" writes p as a low-degree polynomial, and performance depends on the size of the coefficients in this polynomial: quantitatively, security levels are reduced by about 20% when the polynomial has small coefficients.

However, the point of ECC has always been to avoid index calculus in the first place. All of the SafeCurves requirements can be met by "special" primes.

Most ECC standards require "special" primes for efficiency reasons. For example, NIST justifies its prime shape by saying that "modular multiplication can be carried out more efficiently than in general". However, Brainpool prohibits "special" primes for patent reasons [**66**, Section 3.1]:

> The prime p must not be of a special form in order to avoid patented fast arithmetic on the base field.

The context here is that U.S. patent 5159632 had been filed in September 1991 on elliptic-curve cryptography over $\mathbb{F}_p$ "where p is one of a class of numbers such that mod p arithmetic is performed in a processor using only shift and add operations", a common way to reduce modulo primes such as $2^{224} - 2^{96} + 1$. However, a paper by Bender and Castagnoli published in June 1990 [**20**] had already reported an implementation of ECDH using, e.g., the prime $2^{127}+24933$, "which is convenient in computer arithmetic". We are not aware of any reports of enforcement attempts for the patent. The patent expired in 2011.

2.3 Are "random" primes dangerous?

Brainpool's choice of "random" primes p makes arithmetic in $\mathbb{F}_p$ roughly twice as slow. We do not know whether there are examples where this slowdown has motivated protocols using Brainpool to select, e.g., the brainpoolP192t1 curve instead of the brainpoolP256t1 curve; such cases would warrant adding a requirement to SafeCurves to prohibit "random" primes p. The question here is about protocol choices rather than internal implementation choices; if a protocol requires brainpoolP256t1 then interoperability will force each implementation to use brainpoolP256t1.

2.4 Are primes required to be 3 mod 4?

All of the SafeCurves requirements can be met by primes p with $p \equiv 1 \pmod 4$, and by primes p with $p \equiv 3 \pmod 4$.

Brainpool is more restrictive here: it requires $p \equiv 3 \pmod 4$. Brainpool does not claim that this has a security justification but instead claims that it "allows efficient point compression"; see [**66**, Section 3.1]. This boils down to the statement that computing square roots in $\mathbb{F}_p$ is efficient when $p \equiv 3 \pmod 4$, since one can compute a square root of s by computing $r = s^{(p+1)/4}$ and then

checking whether $r^2 = s$. But computing square roots in $\mathbb{F}_p$ has practically identical efficiency when, e.g., $p \equiv 5 \pmod 8$, as in the case of Curve25519: compute $r = s^{(p+3)/8}$, check whether $r^2 = \pm s$, and multiply r by a precomputed $\sqrt{-1} \in \mathbb{F}_p$ if $r^2 = -s$.

We comment—looking ahead to, e.g., Section 9's coverage of invalid-curve attacks—that requiring extra steps can be a security risk if implementations that omit the extra steps *seem* to work while failing in situations that an attacker can trigger. This concern does not apply to the square-root procedure for $p \equiv 5 \pmod 8$: omitting the $\sqrt{-1}$ step for the case $r^2 = -s$ would not pass basic interoperability tests. Also, the difference in performance between the two square-root methods in the previous paragraph is unnoticeable, so the downgrading-security-levels concern from Section 2.3 does not apply.

3 Equations

There are several different ways to express elliptic curves over $\mathbb{F}_p$:

- The **short Weierstrass equation** $y^2 = x^3 + ax + b$, where $4a^3 + 27b^2$ is nonzero in $\mathbb{F}_p$, is an elliptic curve over $\mathbb{F}_p$. Every elliptic curve over $\mathbb{F}_p$ can be converted to a short Weierstrass equation if $p > 3$.
- The **Montgomery equation** $By^2 = x^3 + Ax^2 + x$, where $B(A^2 - 4)$ is nonzero in $\mathbb{F}_p$, is an elliptic curve over $\mathbb{F}_p$. Substituting $x = Bu - A/3$ and $y = Bv$ produces the short Weierstrass equation $v^2 = u^3 + au + b$ where $a = (3 - A^2)/(3B^2)$ and $b = (2A^3 - 9A)/(27B^3)$.
- The **Edwards equation** $x^2 + y^2 = 1 + dx^2y^2$, where $d(1 - d)$ is nonzero in $\mathbb{F}_p$, is an elliptic curve over $\mathbb{F}_p$. Substituting $x = u/v$ and $y = (u-1)/(u+1)$ produces the Montgomery equation $Bv^2 = u^3 + Au^2 + u$ where $A = 2(1 + d)/(1 - d)$ and $B = 4/(1 - d)$.

A 1987 paper by Montgomery [**177**] introduced Montgomery curves. A 2007 paper by Edwards [**95**] introduced Edwards curves in the case that d is a 4th power. For reasons explained in Section 10, SafeCurves requires Edwards curves to be **complete**, i.e., for d to not be a square; we introduced complete Edwards curves in a 2007 paper [**36**].

There are other possibilities, such as Hessian curves, which share some interesting properties with Edwards curves (see, e.g., [**130**]). As far as we know, there are no efforts to deploy specific Hessian curves in ECC. For simplicity, we focus on the three curve shapes specified above.

The **rational points** of a short Weierstrass curve are the pairs (x, y) of elements of $\mathbb{F}_p$ satisfying the equation, together with one extra "point at infinity". The **rational points** of a Montgomery curve are defined the same way. The **rational points** of a complete Edwards curve are the pairs (x, y) of elements of $\mathbb{F}_p$ satisfying the equation; there is no extra "point at infinity". In each of these cases, the set of rational points is written $E(\mathbb{F}_p)$, where E is the curve. There is a standard definition of a group structure on $E(\mathbb{F}_p)$; the neutral element is the

point at infinity for a short Weierstrass curve or a Montgomery curve, or $(0, 1)$ for an Edwards curve.

As a concrete example, Curve25519 is the Montgomery curve $y^2 = x^3 + 486662x^2 + x$ over $\mathbb{F}_p$ with (as mentioned in Section 2) $p = 2^{255} - 19$. Here $B = 1$ and $A = 486662$, so $B(A^2 - 4) \neq 0$ in $\mathbb{F}_p$. As another example, NIST P-224 is the short Weierstrass curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ with $p = 2^{224} - 2^{96} + 1$, $a = -3$, and

$$b = 18958286285566608000408668544493926415504680968679321075787234672564;$$

one can check that $4a^3 + 27b^2 \neq 0$ in $\mathbb{F}_p$. NIST P-256 and brainpoolP256t1 also have the form $y^2 = x^3 - 3x + b$, where

$$b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$$

for NIST P-256 and

$$b = 46214326585032579593829631435610129746736367449296220983687490401182983727876$$

for brainpoolP256t1.

3.1 Are short Weierstrass equations required to have $a = -3$?

The first serious effort to optimize formulas for elliptic-curve group operations was in a 1986 paper [82, Section 4] from D. Chudnovsky and G. Chudnovsky. The paper considered what are now typically called short Weierstrass curves, Jacobi quartics, Jacobi intersections, and Hessian curves. In the case of short Weierstrass curves, the paper considered affine coordinates (x, y), but recommended instead using what are now typically called Jacobian coordinates (X, Y, Z) with $x = X/Z^2$ and $y = Y/Z^3$.

Note that the group operation on an elliptic curve is normally written as addition rather than multiplication. This is also the reason for "nP" in the quotes in Section 1, meaning the sum of n copies of P on the curve.

Discussions of costs frequently refer to the number of multiplications. It is important to be clear about whether these are multiplications in the field or multiplications of scalars n by curve points P. As in the literature, we use **M** to refer to the cost of multiplication in the field.

For Jacobian coordinates, [82] found doubling formulas taking 9**M** plus 1 multiplication by a. Obviously choosing $a = 1$ removes the multiplication by a, but the paper said that it is "even smarter" to choose $a = -3$, where alternative formulas take just 8**M**. The paper also found general addition formulas taking 16**M**. The paper also considered an extension (X, Y, Z, Z^2, Z^3) of Jacobian coordinates, and projective coordinates (X, Y, Z) with $x = X/Z$ and $y = Y/Z$, in both cases reducing general addition from 16**M** to 14**M**, but at the expense of doubling using 9**M** for (X, Y, Z, Z^2, Z^3) or 10**M** for (X, Y, Z).

IEEE P1363 suggests Jacobian coordinates, writing that "Other kinds of projective coordinates exist, but the ones given here provide the fastest arithmetic on elliptic curves. (See [CC87].)" P1363 also presents the $a = -3$ speedup from [82]. Similarly, the NIST curves use $y^2 = x^3 - 3x + b$ "for reasons of efficiency", and Brainpool uses $y^2 = x^3 - 3x + b$ for its "arithmetical advantages". Efficiency is also the rationale stated for various other choices in these standards: for example, NIST takes the "cofactor" (see Sections 7 and 9) to be "as small as possible" for "efficiency reasons". Recall also from Section 2.4 that Brainpool takes $p \equiv 3 \pmod 4$ for "efficient point compression".

If there are cases where the slowdown from using short Weierstrass equations *without* $a = -3$ has triggered a switch to lower security levels, then a requirement for short Weierstrass equations to use $a = -3$ can be justified on security grounds. See the analogous discussion in Section 2.3 of the (larger) speedup from avoiding "random" primes. Note also that one cannot object to an $a = -3$ requirement as potentially allowing an ECDLP attack: such an attack would imply an ECDLP attack against generic a, as explained in [71, Section 6].

4 The rho method

Along with specifying a curve, one specifies a base point G of prime order ℓ on that curve. The basic ECDH protocol is then as follows: user i has a secret key s_i, which is an integer; user i has a corresponding public key s_iG, which is a point on the elliptic curve; users i and j then compute a shared secret s_is_jG, which is a point on the elliptic curve.

Note that, in the original 1976 DH protocol reviewed in Section 1, the base $g \in \mathbb{F}_q^*$ had order $q-1$, which is never prime if $q > 3$ is a prime number (although the order can be prime if q is a power of 2). For a closer analogy to the previous paragraph, one can replace g in DH with g^2, which has prime order ℓ in the case $q - 1 = 2\ell$.

The size of ℓ puts an important limit on the security level. A standard attack [197], called the rho method, breaks ECDLP using, on average, approximately $0.886\sqrt{\ell}$ curve additions. For example, $0.886\sqrt{\ell}$ is approximately $2^{111.8}$ for NIST P-224, approximately $2^{125.8}$ for Curve25519, and approximately $2^{127.8}$ for NIST P-256.

4.1 Can rho finish sooner?

The $0.886\sqrt{\ell}$ cost for the rho method is an average over random choices made by the attacker: the method sometimes finishes more slowly and sometimes finishes more quickly. The success probability of the method after m additions is only about m^2/ℓ for small m. Common recommendations are to choose ℓ so that m^2/ℓ is negligible for any feasible value of m.

In 2013, we established a concrete lower limit for ℓ in SafeCurves as follows: "For example, if ℓ is around 2^{200}, then the success probability is around $1/2^{20}$ after 2^{90} additions. Performing 2^{90} additions in a year with state-of-the-art

chip technology (as of 2013) would require hundreds of gigawatts of power. SafeCurves requires ℓ to be at least 2^{200}." We emphasized that SafeCurves does *not* recommend against larger values of ℓ:

> There are several reasons to use larger ℓ: chip technology can be expected to advance; $1/2^{20}$ is not an acceptable risk level for some users; very few ECC applications will notice the cost of increasing ℓ to, e.g., 2^{250}.

As an illustration of how chip technology has advanced after our estimates of what was possible with 2013 technology, consider the ANTMINER S21 [**57**], a Bitcoin-mining device released at the end of 2023 that uses 3500 watts and carries out 200 terahashes per second, nearly 2^{36} hashes per joule. Each hash uses about 2^{18} bit operations, while Appendix B indicates that a curve addition inside the rho method for, e.g., Curve25519 costs about 2^{20} bit operations, so using similar chip technology should carry out nearly 2^{34} curve additions per joule. Carrying out 2^{90} curve additions thus costs around 2^{56} joules; 2^{56} joules in a year is around 2 gigawatts.

To account for current chip efficiency and expected near-future improvements in chip efficiency, we are updating the SafeCurves lower limit on ℓ from 2^{200} to 2^{220}. We again emphasize that this is not a recommendation against larger values of ℓ: further improvements in chip technology would be unsurprising.

4.2 Can rho take advantage of multiple targets?

Yes. There is a square-root effect for multiple targets: breaking 1000000 keys with the rho method costs only about 1000 times as much as breaking a single key, not 1000000 times as much. See [**156**], [**131**], [**162**], and [**40**].

One can find standards as late as 2001 saying that "the computation of a single elliptic curve discrete logarithm has the effect of revealing a single user's private key. The same effort must be repeated in order to determine another user's private key". The second sentence is incorrect: the effort is reduced for each subsequent key.

For comparison, with secret-key ciphers there is a much worse effect for multiple targets. Breaking a single AES key costs about 2^{128} computations; breaking 1000000 AES keys costs, in total, about 2^{128} computations.

4.3 Does rho find some key more quickly when there are multiple targets?

No. The probability of finding *any* keys within m additions is still only about m^2/ℓ. The first key found will still cost approximately $0.886\sqrt{\ell}$ additions on average.

For comparison, if there are 1000000 AES keys to break, then some key will be found after only $2^{128}/1000000$ computations.

4.4 What is this 0.886? What exactly are the additions?

The 0.886 is actually $\sqrt{\pi/4}$. The additions are "batched affine" short-Weierstrass elliptic-curve additions, each consisting of 5 multiplications mod p, 1 squaring mod p, and an asymptotically negligible amount of extra work. (Short Weierstrass curves are the fastest known curve shapes for batched affine additions. This does not mean that other curves are harder to attack: the attacker converts other curves and the points on them to short Weierstrass form.) The algorithm can be efficiently parallelized and vectorized [187].

4.5 How stable is the security story for rho?

A 1971 paper by Shanks [217] introduced generic square-root DLP attacks. For curves that meet the SafeCurves requirements, the number of additions used in [217] is within a factor 2 of the number of additions used by state-of-the-art ECDLP attacks.

To be more precise, the Shanks method uses $1.5\sqrt{\ell}$ group operations on average. The low-memory rho method from Pollard [197] uses $\sqrt{\pi/2}\sqrt{\ell}$ group operations on average. This formula was used in, e.g., a draft of X9.62 in 1997 and in the Brainpool standard in 2005. There have been three main themes of research since 1971:

- Optimizing the number of additions. The largest change was a $\sqrt{2}$ "negation" speedup, replacing $\sqrt{\pi/2} \approx 1.253$ with $\sqrt{\pi/4} \approx 0.886$. See Wiener [228], Escott [96], and Duursma–Gaudry–Morain [94]; for further analysis see Bos–Kleinjung–Lenstra [60] and our paper [50] with Schwabe.
- Showing that there are no bottlenecks other than the arithmetic required for the additions: in particular, no serious memory use, no serialization, no serious communication, and no serious branching. Pollard [197] introduced the rho method, showing that square-root DLP attacks do not need much memory. Van Oorschot and Wiener [187] (drafts published as early as 1994) showed that a variant of the rho method is parallelizable with negligible communication costs, and [50] showed that a negating variant of the rho method is vectorizable.
- Optimizing the arithmetic inside the additions. See Appendix B.

5 Transfers

A "transfer" converts ECDLP into a "linear algebraic group" DLP. There are several types of transfers (see Section 5.1 for references) for an elliptic-curve group of prime order ℓ over a prime field $\mathbb{F}_p$:

- Additive transfer: applicable when $\ell = p$. The target group is the additive group $\mathbb{F}_p$, where DLP is very easy to solve.

- Degree-1 multiplicative transfer: applicable when ℓ divides $p-1$. The target group is the multiplicative group $\mathbb{F}_p^*$, where DLP is solved in subexponential time by index calculus. The standard estimates are that current index-calculus methods break DLP in $\mathbb{F}_p^*$ at cost below 2^{128} for p up to roughly 2^{3000}.
- Degree-2 multiplicative transfer: applicable when ℓ divides p^2-1. The target group is the multiplicative group $\mathbb{F}_{p^2}^*$, where DLP is solved in subexponential time by index calculus. The standard estimates are that current index-calculus methods break DLP in $\mathbb{F}_{p^2}^*$ at cost below 2^{128} for p up to roughly 2^{1500}.
- Degree-3 multiplicative transfer: applicable when ℓ divides p^3-1. The target group is the multiplicative group $\mathbb{F}_{p^3}^*$, where DLP is solved in subexponential time by index calculus. The standard estimates are that current index-calculus methods break DLP in $\mathbb{F}_{p^3}^*$ at cost below 2^{128} for p up to roughly 2^{1000}.
- Et cetera.

The minimum possible multiplicative-transfer degree for a particular elliptic-curve group is called the **embedding degree** of that group. Standards vary in the requirements they place upon the embedding degree:

- SEC1 [**203**] requires the embedding degree to be at least 20.
- X9.62 [**8**] requires the embedding degree to be at least 20.
- P1363 [**134**] puts variable requirements upon the embedding degree, depending on the size of p, but never requires it to be more than 30.
- Brainpool [**66**] requires the embedding degree to be at least $(\ell-1)/100$.

The SEC/X9.62/P1363 approach is risky: there is a long history of improvements to index calculus, and the point of ECC has always been to avoid index calculus. The Brainpool approach is clearly overkill, but is also non-controversial, since it rules out only a small fraction of curves [**17**].

SafeCurves takes the overkill approach. Pairing-based cryptography requires the risky approach, but pairing-based cryptography is outside the scope of SafeCurves.

5.1 How stable is the security story for transfers?

All of these transfers have been known since the 1990s. Multiplicative transfers were introduced by Menezes–Okamoto–Vanstone [**172**], Frey–Rück [**105**], and Semaev [**214**], and are often called the "MOV attack". Additive transfers were introduced by Semaev [**215**], Satoh–Araki [**209**], and Smart [**218**], and are often called the "Smart-ASS attack".

6 CM field discriminants

The number of rational points $\#E(\mathbb{F}_p)$ on an elliptic curve E over $\mathbb{F}_p$ is $p+1-t$ where t is the **trace** of E. Hasse's theorem states that t is between $-2\sqrt{p}$ and $2\sqrt{p}$. The prime order ℓ of the base point from Section 4 is a divisor of $p+1-t$.

If s^2 is the largest square dividing $t^2 - 4p$ then $(t^2 - 4p)/s^2$ is a squarefree negative integer. Define D as $(t^2 - 4p)/s^2$ if $(t^2 - 4p)/s^2 \equiv 1 \pmod 4$, otherwise as $4(t^2 - 4p)/s^2$. SafeCurves requires the absolute value of this **complex-multiplication field discriminant** D to be larger than 2^{100}. We are updating this to 2^{110} to keep pace with Section 4.1.

As in Section 5 (and unlike Section 4), this requirement is satisfied by the vast majority of elliptic curves. Looking beyond the scope of SafeCurves, we note the reasons that the literature sometimes intentionally chooses curves with small D: these curves make $\#E(\mathbb{F}_p)$ easier to compute (but $\#E(\mathbb{F}_p)$ does not take long to compute in any case), give a speedup in computing nP [**112**] at the expense of more complicated computation, and play a critical role in pairing-based cryptography.

6.1 How do I verify the trace?

Verifying that the base point has order ℓ guarantees that the curve cardinality $p+1-t$ is a multiple of ℓ. Typically ℓ is above $4\sqrt{p}$, so there is only one multiple of ℓ between $p+1-2\sqrt{p}$ and $p+1+2\sqrt{p}$; this multiple must be $p+1-t$.

In more detail, if all six of the following checks pass, then base point P has prime order ℓ and the curve trace is t: check that (1) ℓ is prime; (2) $\ell^2 > 16p$; (3) ℓ divides $p+1-t$; (4) $t^2 < 4p$; (5) P is not the neutral element; and (6) ℓP is the neutral element.

Proof: The order of P is a divisor of ℓ by condition 6, but ℓ is prime by condition 1, so the order of P is 1 or ℓ, but the order of P is larger than 1 by condition 5, so the order of P is exactly the prime ℓ as claimed. The order of P also divides $\#E(\mathbb{F}_p)$, since the order of each element divides the cardinality of the group. Write $t' = p+1-\#E(\mathbb{F}_p)$; then ℓ divides $p+1-t'$. Now $-2\sqrt{p} \le t' \le 2\sqrt{p}$ by Hasse's theorem, while $-2\sqrt{p} < t < 2\sqrt{p}$ by condition 4, so $|t-t'| < 4\sqrt{p}$. Also $4\sqrt{p} < \ell$ by condition 2, so $|t-t'| < \ell$. But ℓ divides $p+1-t$ by condition 3, so ℓ divides $t-t'$. This forces $t-t' = 0$; i.e., the curve trace t' is t as claimed.

6.2 Is ECDLP broken for curves with small $|D|$?

Slightly. Specifically, starting from the rho method (see Section 4), one can save time for some curves where $|D|$ is very small, using fast "endomorphisms" derived from D [**111**].

This is not a complete break. The limits of these speedups are reasonably well understood, and the literature does not indicate any mechanism that could allow further speedups for small $|D|$, except when pairings allow the transfers covered in Section 5. It is conceivable that there are much better attacks against the occasional curves with small $|D|$, but, in the opposite direction, it is conceivable that there are much better attacks against the usual curves with large $|D|$. What is clear is that the security story is more complicated for small $|D|$; SafeCurves therefore requires large $|D|$.

Brainpool contains a related requirement: the **class number**, a quantity related to D, is required to be larger than 1000000. The generalized Riemann

hypothesis (a standard conjecture in number theory, backed by extensive evidence) implies that the class number is not far from the square root of $|D|$; it is thus reasonably clear that the Brainpool requirement is much weaker than the SafeCurves lower limit on $|D|$. With some computation one can compute exact class numbers, and with less computation one can verify the Brainpool class-number condition, but this has not been incorporated into SafeCurves.

7 Rigidity

There are documented instances (see, e.g., [219], [49], [194], and [26, Section 3.6]), and many more suspected instances, of standards being manipulated by attackers. This raises the question of how users of standard curves can be assured that the curves were not generated to be weak.

The SafeCurves criteria simplify the ECC security story by requiring prime fields (see Section 2). This still leaves various security dangers such as transfers (see Section 5) and invalid-curve attacks (see Section 9 below), but the SafeCurves tables check for these dangers in a publicly verifiable way. There is still a potential lack of assurance in the following corner case:

- public ECC cryptanalysis might have missed an attack that applies to a small fraction of curves,
- an attacker might have figured out this attack, and
- the attacker might have manipulated the choices of standard curves to be vulnerable to this secret attack.

Most ECC standards include mechanisms that supposedly block the third step in this scenario. However, further analysis has pointed out flaws in these mechanisms. This section highlights three examples of algorithms that—*if* they are given a small class of weak curves as input—manipulate curve choices to land in that class, despite claimed barriers to this manipulation.

7.1 Manipulation algorithm 1: seed search

The possibility of attackers manipulating standard curve choices was raised in the late 1990s, when NSA volunteered to "contribute" elliptic curves to the committee producing ANSI X9.62 [8]. NSA did in fact end up producing various elliptic curves later standardized by ANSI X9.62, SEC 2 [203], and NIST FIPS 186-2 [220]; these curves, the "NIST curves", were subsequently deployed in many applications.

(As an explanation for NSA's involvement, Koblitz and Menezes wrote the following [153]: "In 1997, counting the number of points on a random elliptic curve was still a formidable challenge." However, a 1998 implementation paper [141] reported counting points on 3569 curves with $p = 2^{240} + 115$, with early aborts, in a total of 52 hours on a 300MHz Pentium II, under a minute per curve, finding 16 examples of prime-order curves.)

In response to NSA's contributions, ANSI X9.62 developed "a method for selecting an elliptic curve *verifiably* at random", and a procedure to "verify that a given elliptic curve was indeed generated at random". ANSI X9.62 even claims that this procedure "serves as proof (under the assumption that SHA-1 cannot be inverted) that the parameters were indeed generated at random". However, this procedure does *not* verify randomness; it verifies only that the curve coefficients were produced as hash output. The claimed "proof" is nonexistent.

Concretely, NIST P-256 is a curve of the form $y^2 = x^3 - 3x + H(s)$, where s is a large public "seed" and H is a hash function. In 1999, shortly after the NIST curves were announced, Scott [212] pointed out that the curves were not, in fact, verifiably random:

> Now if the idea is to increase our confidence that these curves are therefore completely randomly selected from the vast number of possible elliptic curves and hence likely to be secure, I think this process fails. The underlying assumption is that the vast majority of curves are "good". Consider now the possibility that one in a million of all curves have an exploitable structure that "they" know about, but we don't.. Then "they" simply generate a million random seeds until they find one that generates one of "their" curves. Then they get us to use them. And remember the standard paranoia assumptions apply - "they" have computing power way beyond what we can muster. So maybe that could be 1 billion.

Scott recommended generating curve coefficients from digits of π as an alternative, and concluded his posting as follows: "So, sigh, why didn't they do it that way? Do they want to be distrusted?"

In 2000, SEC 2 version 1.0 copied the curves that NSA had produced for NIST, copied the claim that the curves were "chosen verifiably at random", and specifically claimed that the curves were chosen "by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found". This claim might be correct, but is certainly not verifiable.

7.2 Manipulation algorithm 2: curve-generator search

In 2005, Brainpool identified the lack of explanation of the NSA/NIST curve seeds as a "major issue" [66, page 2]. Instead of claiming to generate seeds at random, Brainpool specified a deterministic procedure to generate seeds "in a systematic and comprehensive way", reminiscent of Scott's suggestion to use digits of π.

In a May 2013 talk [42], we again raised the possibility of NSA having searched through a billion curves. The main focus of the talk—and of the SafeCurves web site [43], which we announced in October 2013—was instead implementation issues, but we did include a "rigidity" criterion in SafeCurves. This criterion prohibits the NIST curves as "manipulatable": the process "has

a large unexplained input, giving the curve generator a large space of curves to choose from". The criterion allows the Brainpool curves because the curve generators do not have "many bits of control", but we also pointed out that the Brainpool seed-generation mechanism was only partially explained:

> Why SHA-1 instead of, e.g., RIPEMD-160 or SHA-256? Why use 160 bits of hash input independently of the curve size? Why pi and e instead of, e.g., sqrt(2) and sqrt(3)? Why handle separate key sizes by more digits of pi and e instead of hash derivation? Why counter mode instead of, e.g., OFB? Why use overlapping counters for A and B (producing the repeated 26DC5C6CE94A4B44F330B5D9)? Why not derive separate seeds for A and B?

Together with Chou, Chuengsatiansup, Hülsing, Lambooij, Niederhagen, and van Vredendaal, we then refined the SafeCurves analysis, developing models for criteria applied to accept curves and quantifying the number of curves accepted by each model. Our paper [**29**] showed how to generate more than a billion different Brainpool-like curves, each claiming to be "verifiably pseudorandom". (We also computed more than a million such curves.) In other words, the Brainpool approach—despite obviously being more constrained than the NIST approach—would still have supported malicious generation of a curve having a one-in-a-billion weakness, perhaps indicating that the SafeCurves rigidity criterion should be strengthened to disallow this approach. We also showed how to generate hundreds of thousands of curves meeting a newer criterion of being "verifiably deterministic", and how to generate hundreds of Curve25519-like curves, each being the fastest curve meeting specified security criteria.

7.3 The NIST curves, revisited

NSA's Jerry Solinas—who had supplied the curves standardized by ANSI, NIST, etc.—sent email after the first version of [**29**] went online. Given public interest in these issues, we have decided to disclose the email exchange; see Appendix C. Solinas claimed that NSA had "built all the seeds via hashing (SHA-1, I think) from the ASCII representation of a humorous message ... I believe there was a counter rather than multiple hashing, but I don't know details. The message was along the lines of 'Give Bob and Jerry a raise' or 'Bob and Jerry rule' or something like that".

Note that this claim is incompatible with the claims of verifiable randomness reviewed in Section 7.1, such as the claim that the curves were generated "by repeatedly selecting a random seed and counting the number of points on the corresponding curve until appropriate parameters were found". The reader understands "random" to mean uniformly and independently at random, not a meaningful message accompanied by a counter.

We tried hashing many inputs along the lines sketched above, varying details much as in Section 7.2, hoping to find the original seed. After a while, we gave up and moved the computers to more productive projects. We have heard about more recent efforts to reconstruct the seed, apparently none succeeding yet.

Solinas claimed that "we could prove our innocence by disclosing the details, if only we could remember them". But success in recovering the seed would not provide any evidence of innocence. There are far more than a billion choices of plausible, efficiently enumeratable messages.[4] A malicious search hashing many seeds to target a one-in-a-billion weakness does not care whether the seeds were generated randomly or generated as meaningful messages.

7.4 Manipulation algorithm 3: isogenies

Koblitz and Menezes claimed in 2015 [**153**, 21 October 2015 eprint version, Section 3.1] that, since there are $p^{1+o(1)}$ isomorphism classes of elliptic curves over $\mathbb{F}_p$, an attacker knowing a class of weak curves and searching (say) $p^{1/4+o(1)}$ seeds would be able to find a curve in the class only if the class has size at least $p^{3/4+o(1)}$, a "huge class of weak curves".

We pointed out in 2017 that this argument is incorrect, even for the extreme case of "back door" weaknesses. Our curve-manipulation algorithm works as follows (see also [**67**] for portions of this structure):

- Write down $p^{1/4+o(1)}$ back-door keys. Map each key to a curve, and count the number of points on each curve. (We assume that the number of points on a backdoored curve looks random. One out of every $p^{o(1)}$ curves will have prime order.)
- Write down $p^{1/4+o(1)}$ seeds. Map each seed to a curve, and count the number of points on each curve.
- There is a good chance of a collision in the number of points, since this number is in an interval of length only $4p^{1/2}$ by Hasse's theorem.
- Given a back-door key and a seed producing the same number of points, find an efficient isogeny between the two curves as follows: compute $p^{1/4+o(1)}$ curves efficiently isogenous to the first curve; compute $p^{1/4+o(1)}$ curves efficiently isogenous to the second curve; find a collision.

This algorithm takes $p^{1/4+o(1)}$ operations to find a curve in a class of size only $p^{1/4+o(1)}$, far smaller than the $p^{3/4+o(1)}$ claimed in [**153**]. The memory consumption in the algorithm can easily be eliminated if the pool of back-door

[4] Consider, e.g., a simple request for money: "Bob" can instead be written "Bob Reiter" or "Robert Reiter" or "Robert Reiter Jr." or "Robert Reiter, Jr." or "Robert W. Reiter" or "Robert W. Reiter Jr." or "Robert W. Reiter, Jr."; "Dr." can be inserted in front of the name; if there were 20 people in the office with different names then there were 1330 choices of 2 or 3 names in alphabetical order; the request can end with an exclamation mark, a period, or no punctuation; the request can be all uppercase, all lowercase, or normal; a counter can be before or after the sentence; the counter can be separated by a space, a slash, a colon, or nothing; the counter can be zero-padded to 6 digits, or unpadded; the counter can be decimal, `0o` octal, `0x` hex, or `0X` hex; the request can start with "please" or "we must" or nothing; the request can end with "today" or "right now" or "this year" or nothing; the request can be for a "raise" or a "bonus"; this can be "big" or "large" or "giant" or "massive" or "enormous"; etc.

keys is large enough (as it has to be for "nobody but us" back doors) and if the pool of seeds is large enough (as it is for the NIST curves).

Koblitz and Menezes responded that they estimate at least 2^{86} bit operations for this procedure for $p \approx 2^{256}$, "which almost certainly was beyond the NSA's capacity in 1997".

To put 2^{86} bit operations in perspective, observe that Bitcoin currently carries out 2^{87} bit operations every second. (See [58], which indicates that Bitcoin has reached 600 million terahashes per second; and recall that each hash uses about 2^{18} bit operations.) The question is whether this volume of computation would have taken, e.g., a year for NSA in 1997.

Appendix D uses information about Intel's mass-produced chips in 1997 to estimate the volume of chips that would have carried out 2^{86} bit operations in a year, and concludes that this would have been only about 10% of NSA's budget at the time, according to public information regarding that budget.

Of course, if records show that NSA was responding to an unpredictable curve-generation challenge in only M months, then an attack would have had to complete within M months. On the other hand, presumably more attention to the attack details would reduce the number of bit operations.

7.5 Attack discovery

The real question in [153] is whether it is plausible that NSA in 1997 knew curve weaknesses that are not known to the public today. Regarding a class of, e.g., 2^{209} weak curves, [153] says that it is "highly unlikely that such a large family of weak elliptic curves would have escaped detection by the cryptographic research community from 1997 to the present".

We do not see why the size of a weak-key class is relevant to the question of whether the weakness is detected by the public. The currently known classes of curves for which ECDLP has been broken range from the tiny class of curves subject to additive transfers (see Section 5) to the frequent curves for which the group order factors into small primes. Weak curves are detected by study of potential avenues of attack, not by Geiger counters.

If there are many smart people in public actively searching for better ECDLP attacks then the passage of time would suggest that no such attacks exist. However, public cryptanalysts with relevant knowledge are generally busy working on other topics, such as isogeny-based cryptography; so one would expect an elliptic-curve attack to appear only if it happens to be a spinoff of those other topics, as in [109].

7.6 Isn't it safest to choose cryptographic parameters at random?

Cryptographic *keys* lose security when they do not have enough randomness. There is a common confusion between public *parameters* and public *keys*, creating a common myth that public *parameters* lose security unless they are as random as possible.

The literature contains many counterexamples to this myth. For example, there are known attacks [90] that significantly reduce the security level of random genus-3 curves, but the attacks do not apply to specially structured genus-3 curves, namely *hyperelliptic* curves. As another example, ECC takes only unusual curves whose group orders have very large prime divisors, because uniform random curves are much less secure than these unusual curves. See [151, Section 11] for more subtle examples.

One should not conclude that uniform random parameters are necessarily bad: there are also examples where adding randomness to parameters is good. To see whether randomness is good or bad for the parameters of any particular system, one needs to study the details of attacks against that system.

All curves that meet the SafeCurves criteria are protected against all published attacks, use the most conservative bounds to stay far away from those attacks, and eliminate structure that has raised concerns about potential attacks. The criteria are computer-verified, with full details presented on our web page [43] to support third-party verification. It is conceivable that some of these curves are vulnerable to an attack that is not publicly known, but there is no basis for guessing whether any particular curve will be more or less vulnerable to attack than a random curve.

ECC users can reasonably choose their own random curves to protect against multiple-target rho attacks; see Section 4.2. However, giving a random curve to each user also has several obvious costs, whereas moving to a larger shared curve has lower costs and makes the same attacks even more expensive. This is why essentially all ECC applications use shared curves.

7.7 What about rigid choices of subgroups?

For each curve in the SafeCurves tables, the order ℓ of the specified subgroup of the group of rational points is prime and larger than $\sqrt{p}+1$. A curve cannot have two different subgroups meeting this requirement.

7.8 What about rigid choices of base points?

For each curve in the SafeCurves tables, the specified base point is a generator of the specified subgroup. The SafeCurves criteria do not place restrictions on the choice of this base point. If there is a "weak" base point W allowing easy computations of discrete logarithms, then ECDLP is weak for every base point: an attacker can compute $\log_P Q$ as the ratio of $\log_W Q$ and $\log_W P$ modulo ℓ. Typical ECC protocols, such as signatures, are designed to be secure for all choices of base point.

There are some protocols where base-point rigidity is important. For example, a "random" ECDLP challenge, computing the discrete logarithm of Q base P, could have a back door for the challenge creator. Certicom's ECDLP challenges use rigid generators P and Q of the subgroup to prevent Certicom from choosing the discrete logarithm in advance. As another example, the CurveBall attack in 2020 [173] broke signature verification in Microsoft Windows CryptoAPI,

allowing attackers to freely sign malicious executables under Microsoft's key, by exploiting the fact that CryptoAPI allowed certificates to provide their own base points.

For some curves, the specified base point is chosen rigidly. The usual choice is the generator with smallest possible x-coordinate for short Weierstrass curves or Montgomery curves, or smallest possible y-coordinate for Edwards curves. The reason for x vs. y here is that $y(-P) = y(P)$ for Edwards, allowing y as a ladder coordinate (see Section 8), while $x(-P) = x(P)$ for the others, allowing x as a ladder coordinate.

8 Ladders

This section focuses on the most important computation in ECC: namely, single-scalar variable-base-point multiplication. This means computing nP, given an integer n and a curve point P. This is what happens in each shared-secret computation in ECDH, when user i computes $s_i(s_jG)$ using their secret s_i and user j's public key s_jG before starting to communicate with j.

Given the emphasis on efficiency in the turn-of-the-century ECC standards (see the quotes in Section 3.1), it is puzzling that those standards chose to use short Weierstrass curves. This choice did not provide the fastest arithmetic known on elliptic curves; it was already outperformed by other options in [**82**] and [**177**], the fastest being the "Montgomery ladder" introduced in [**177**]. The Montgomery ladder is also a much *simpler* way to carry out ECDH computations, and naturally avoids various classes of security problems that we have seen repeatedly appearing in ECDH software.

This section begins by reviewing scalar-multiplication algorithms for short Weierstrass curves and for Montgomery curves, and then uses timing attacks to introduce the idea of different curve shapes having different chances of producing security problems. Sections 9 and 10 explain more ways that short Weierstrass curves produce security problems avoided by better choices of curves.

8.1 Scalar multiplication on short Weierstrass curves

As an example of how the literature suggests computing nP on a short Weierstrass curve, we quote the algorithm presented in the P1363 standard [**134**, Annex A.10.3, "Elliptic Scalar Multiplication"]:

1. If $n = 0$, then output O and stop.
2. If $n < 0$, then set $Q \leftarrow (-P)$ and $k \leftarrow (-n)$; else set $Q \leftarrow P$ and $k \leftarrow n$.
3. Let $h_l h_{l-1} \ldots h_1 h_0$ be the binary representation of $3k$, where the most significant bit h_l is 1.
4. Let $k_l k_{l-1} \ldots k_1 k_0$ be the binary representation of k.
5. Set $S \leftarrow Q$.
6. For i from $l - 1$ downto 1 do

Set $S \leftarrow 2S$.
If $h_i = 1$ and $k_i = 0$, then compute $S \leftarrow S + Q$ via A.10.1 or A.10.2.
If $h_i = 0$ and $k_i = 1$, then compute $S \leftarrow S - Q$ via A.10.1 or A.10.2.

7. Output S.

We focus on the case $n > 0$, where the first two steps of the algorithm can be omitted. The algorithm builds an "addition-subtraction chain" for n using approximately $\log_2 n$ doublings and $(\log_2 n)/3$ further group operations, each of those group operations being an addition or a subtraction.

The standard also cites sources presenting addition-subtraction chains that are more complicated but more efficient, replacing $(\log_2 n)/3$ with about $(\log_2 n)/5$ for typical sizes of n. These chains are organized as a preliminary computation of a table containing, e.g., $3P, 5P, \ldots, 15P$, and then a series of doublings with occasional additions or subtractions interspersed. The costs shown below assume that these faster chains are used.

The implementor then needs to plug in formulas for curve doubling ("$2S$"), addition ("$S+Q$"), and subtraction ("$S-Q$"), where Q is one of the precomputed small multiples of P from the table. For short Weierstrass curves with $a = -3$ in Jacobian coordinates using the 1986 Chudnovsky–Chudnovsky formulas from [82], each doubling costs $8\mathbf{M}$, and each addition or subtraction costs $16\mathbf{M}$, for a total of about $8\mathbf{M} + 16\mathbf{M}/5 = 11.2\mathbf{M}$ per bit of n. There is also an inversion at the end of the computation to replace Jacobian coordinates (X, Y, Z) with affine coordinates $(x, y) = (X/Z^2, Y/Z^3)$ for communication.

A simple inversion method costs slightly over $1\mathbf{M}$ per bit of p, which typically means slightly over $1\mathbf{M}$ per bit of n. More sophisticated inversion methods (see, e.g., [53]) cost less. Sometimes inversions can be batched, reducing the cost very close to $0\mathbf{M}$ per bit. Using mixed coordinates from [83], which switches between coordinate systems within a scalar multiplication, reduces the costs per additon to $14\mathbf{M}$, for a cost per bit of $8\mathbf{M} + 14\mathbf{M}/5 = 10.8\mathbf{M}$ or, at the expense of an extra inversion in $\mathbb{F}_q$, of $8\mathbf{M} + 11\mathbf{M}/5 = 10.2\mathbf{M}$. See generally [38] and [37] for different coordinate systems and combinations.

8.2 Scalar multiplication on Montgomery curves

For a Montgomery curve $By^2 = x^3 + Ax^2 + x$ in Montgomery coordinates using the Montgomery ladder, computation of nP follows a much simpler pattern of one doubling and one "differential addition" for each bit of n. Differential addition means computing $P + Q$ given P, Q, and $P - Q$.

Each point has just two coordinates (X, Z) with $x = X/Z$; the y-coordinate is not used. The doubling and differential addition together cost just $9\mathbf{M}$ plus a multiplication by $(A + 2)/4$. Montgomery curves are normally chosen with $(A+2)/4$ being small, so the overall cost is just $9\mathbf{M}$ per bit of n, plus an inversion at the end of the computation to convert (X, Z) to $x = X/Z$ for communication.

```
def montgomery(x1,n):
  A = 486662
  x2,z2,x3,z3 = 1,0,x1,1
  for i in reversed(range(255)):
    ni = bit(n,i)
    x2,x3 = cswap(x2,x3,ni)
    z2,z3 = cswap(z2,z3,ni)
    x3,z3 = 4*(x2*x3-z2*z3)**2,4*x1*(x2*z3-z2*x3)**2
    x2,z2 = (x2**2-z2**2)**2,4*x2*z2*(x2**2+A*x2*z2+z2**2)
    x3,z3 = x3%p,z3%p
    x2,z2 = x2%p,z2%p
    x2,x3 = cswap(x2,x3,ni)
    z2,z3 = cswap(z2,z3,ni)
  return (x2*pow(z2,p-2,p))%p
```

Fig. 8.2.1. Python code for the Montgomery ladder for Curve25519. The `bit` function extracts the coefficient of 2^i in n. The `cswap` function is a conditional swap, returning its first two inputs in the same order or reversed order depending on whether the third input is 0 or 1. Warning: integer arithmetic in Python takes variable time.

This is faster than any of the scalar-multiplication methods summarized in Section 8.1; in particular, it is about 1.5 times as fast as the method quoted from P1363. A more detailed cost analysis would consider, e.g., the costs of additions and subtractions in $\mathbb{F}_p$, the speedups from squarings in $\mathbb{F}_p$ being faster than general multiplications in $\mathbb{F}_p$, and the costs of communicating (x, y) for short Weierstrass curves vs. just x for Montgomery curves.

The 2006 paper [**23**] introducing X25519 (ECDH using Curve25519 with a Montgomery ladder)[5] presented X25519 software for various platforms more than twice as fast as previous results for NIST P-256 ECDH, and attributed the speedup partly to the curve choice. Subsequent work has consistently shown X25519 outperforming NIST P-256 ECDH; see Appendix E for examples of current speeds. The speedup is not entirely from the Montgomery ladder—for example, it is also affected by reductions mod $2^{255} - 19$ being easier than reductions mod $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$—but the speed of the Montgomery ladder certainly plays an important role.

What is even more remarkable than the speed of the Montgomery ladder is its simplicity. Figure 8.2.1, copied from the Python test suite in the lib25519 [**51**] software library for Curve25519, displays the Montgomery ladder in the case $A = 486662$ with a 255-bit n—except that the polynomials in the middle such as $x_2x_3 - z_2z_3$ are written for maximum conciseness, skipping the optimized computations of those polynomials from [**177**, page 261, "costs drop"]. These concise formulas were found independently by Chudnovsky–Chudnovsky [**82**, formula (4.19)]; but credit is normally assigned to Montgomery, who found the

[5] [**23**] used "Curve25519" to refer to X25519 rather than to the curve. X25519 was a subsequent renaming to allow separate names for the curve and the protocol.

concise formulas, the optimizations, and a simple statement of the curve shape. (As an example of these optimizations, the reader is invited to consider the two quantities $(x_2 - z_2)(x_3 + z_3) \pm (x_2 + z_2)(x_3 - z_3)$.)

Figure 8.2.1 has similar length to the much slower scalar-multiplication method quoted at the beginning of Section 8.1—but Figure 8.2.1 includes all of the necessary elliptic-curve computations! Furthermore, as we will see, the approach in Section 8.1 naturally leads to various security problems avoided by the Montgomery ladder.

8.3 Timing attacks

Care is required in computing the `cswap` operations in Figure 8.2.1. The obvious way to conditionally swap x_2 and x_3 if $n_i = 1$ is to write code such as

```
if ni == 1: x2,x3 = x3,x2
```

but this will take more time if $n_i = 1$. One then expects sufficiently detailed measurements of the total time of the computation to detect the number of i for which $n_i = 1$. More subtly, interactions with the timings of other programs (see, e.g., [10]) can leak each n_i. A typical way to proactively eliminate these timing leaks is to replace each secret-dependent branch with arithmetic:

```
x2,x3 = x2+ni*(x3-x2),x3-ni*(x3-x2)
```

Further work is then required to ensure that each lower-level operation, such as field addition, takes time independent of the inputs.

There are much larger timing leaks in the P1363 scalar-multiplication algorithm quoted in Section 8.1. For example, the case $(h_i, k_i) = (1, 0)$ triggers an entire curve-addition operation $S \leftarrow S + Q$, a much larger operation than swapping x_2 with x_3. Larger leaks tend to be exploitable in more situations—the timing information can be detected through more noise—and tend to be more annoying to fix. Regarding fixes in this case, one way to replace branches with arithmetic for both $S \leftarrow S + Q$ and $S \leftarrow S - Q$ is to

- conditionally select between Q and $-Q$,
- add the resulting $\pm Q$ to S, and
- conditionally select between S and $S \pm Q$;

also, as before, one needs to make sure that the conditional operations and lower-level arithmetic operations take constant time. Protection here takes more work than in the Montgomery ladder, with more steps that can go wrong. Furthermore, the resulting algorithm takes one curve addition and one doubling for a total of $24\mathbf{M}$ per bit, making it more likely that implementors will switch to the more complicated chains mentioned in Section 8.1, using tables to reduce the number of additions. Those tables, in turn, open up further attacks that exploit timing variations in table lookups; see, e.g., the 2009 paper [74] from Brumley and Hakala demonstrating recovery of elliptic-curve keys from OpenSSL.

We are not saying that implementations *always* get this wrong. It is possible to write software for addition-subtraction chains, even chains at the aforementioned $(\log_2 n)/5$ level of performance, while avoiding secret-dependent conditional branches and secret-dependent table indices. What we are saying is that implementations are *more likely* to have security failures for short Weierstrass curves than for Montgomery curves: the natural pursuit of simplicity—and, in many cases, speed—pushes implementations farther away from security for short Weierstrass curves than for Montgomery curves. See Sections 9 and 10 for further examples of this tension.

8.4 The SafeCurves ladder criterion

The "ladder" criterion in SafeCurves is defined as follows:

> SafeCurves requires curves to support simple, fast, constant-time single-coordinate single-scalar multiplication, avoiding conflicts between simplicity, efficiency, and security. This is not a requirement specifically to use Montgomery curves: there are other types of curves that support simple, fast, constant-time ladders. "Fast" means that implementations of scalar multiplication for the same curve cannot be much faster, and "simple" means that reasonably fast implementations of scalar multiplication for the same curve cannot be much more concise. At this time there are no examples close enough to the edge to warrant quantification of "much".

In this criterion, "single-coordinate" refers to scalar multiplication taking just one coordinate as input and producing just one coordinate as output. "Ladder" refers to the structure used in Figure 8.2.1, with a doubling and differential addition of two points for each bit of n.

Beware that there are some papers erroneously referring to arbitrary ladders as "Montgomery ladders". The general ladder structure is much older than the Montgomery ladder. See, e.g., the discussion of the Lucas ladder in our paper [**45**, Section 4.2.1]; Montgomery's work started with the Lucas ladder.

8.5 Variable-length ladders

Figure 8.2.1 is an example of a constant-length ladder: it initializes the starting variables `x2,z2,x3,z3` so that the algorithm correctly computes nP for any integer n with $0 \leq n < 2^{255}$, always taking exactly 255 iterations.

One can find literature presenting ladders where the number of iterations is instead a variable i, namely the smallest integer $i \geq 0$ such that $n < 2^i$. This does not provide a noticeable speedup or code simplification, but implementors might end up using it simply because it appears in the literature, and then timing information leaks i. This was used in a 2011 timing attack [**75**] to extract secret keys remotely from OpenSSL's implementation of binary-field ECC.

We repeat Section 1's comment about the word "most": better curve choices make ECC implementation failures less likely and are the focus of SafeCurves,

but one needs to take further steps to eliminate the remaining failures. In particular, to prevent problems in case implementors use fixed-length ladders, X25519 uses "a fixed position for the leading 1 in the secret key", as mentioned in the Curve25519 paper [23] in 2006.

8.6 Isomorphism (and birational equivalence)

As noted in Section 3, a simple change of variables converts a Montgomery curve into a short Weierstrass curve. This means that ECDH implementations for a Montgomery curve *can* use the approach from Section 8.1 rather than the much nicer Montgomery ladder. A library that already has software for the NIST curves might be tempted to handle a Montgomery curve this way. We do not claim that using Montgomery curves guarantees that all implementations are secure.

More interestingly, it is sometimes possible to invert this change of variables, converting a short Weierstrass curve $y^2 = x^3 + ax + b$ into a Montgomery curve as follows. Find r satisfying $r^3 + ar + b = 0$. Find s satisfying $s^2 = 3r^2 + a$. Define $u = (x - r)/s$, $B = 1/s^3$, and $A = 3r/s$. Then $By^2 = u^3 + Au^2 + u$. One can perform x-coordinate scalar multiplication on $y^2 = x^3 + ax + b$ by converting x to u, performing u-coordinate scalar multiplication on $By^2 = u^3 + Au^2 + u$ with the Montgomery ladder, and converting back.

The reason this does not always work is that, for the majority of curves, the field $\mathbb{F}_p$ does not contain suitable elements r and s. One can work around this by replacing $\mathbb{F}_p$ with an extension field, but this requires more complicated field operations inside scalar multiplication.

To be more precise, out of all isomorphism classes of elliptic curves over $\mathbb{F}_p$, the fraction that can be written as Montgomery curves is about $5/12$ if $p \equiv 1 \pmod 4$, and about $3/8$ if $p \equiv 3 \pmod 4$; see, e.g., [27]. A Montgomery curve E always has $\#E(\mathbb{F}_p) \equiv 0 \pmod 4$, so curves E over $\mathbb{F}_p$ for which $\#E(\mathbb{F}_p)$ is a prime number, or 2 times an odd prime number, can never be converted to Montgomery curves over $\mathbb{F}_p$.

8.7 More ladders

Instead of trying to convert a short Weierstrass curve to a Montgomery curve, one can build a ladder directly on a short Weierstrass curve.

Every curve has a ladder. In the first ECC paper in 1986 [174, page 425, fourth paragraph], Miller commented that "only the x-coordinate needs to be transmitted" for ECDH on short Weierstrass curves $y^2 = x^3 + ax + b$, since "the x-coordinate of a multiple depends only on the x-coordinate of the original point"; one can convert the formulas given in [174] (attributed there to earlier sources) into a constant-time ladder.

We emphasize that merely having a ladder is not sufficient to meet the SafeCurves ladder criterion. The ladder has to also be "fast", meaning that "implementations of scalar multiplication for the same curve cannot be much

faster", and "simple", meaning that "reasonably fast implementations of scalar multiplication for the same curve cannot be much more concise". This criterion is not simply asking whether a ladder *exists*; it is asking whether natural implementation incentives will lead to the ladder being *used*, as in the case of the Montgomery ladder.

In 2002, Brier and Joye [70] reported 19**M** per bit for a constant-time x-coordinate ladder applicable to every short Weierstrass curve $y^2 = x^3 + ax + b$. Scalar multiplication for the same curves as in Section 8.1 is much faster than the Brier–Joye ladder, so this ladder does not qualify as "fast". We commented on the SafeCurves web site that this ladder is an example of a "conflict between efficiency and security".

There has been more work on ladders since then. The latest news is a 2020 paper from Hamburg [124] reporting 11**M** per bit for an x-coordinate ladder. For comparison, recall that the best speed from Section 8.1 was 10.2**M** per bit. The performance gap turns out to be larger when one accounts for squarings being faster than general multiplications in $\mathbb{F}_p$, but one might still ask whether the word "much" now needs be quantified in the definition of "fast". Independently of that question, there is a different reason that we do not expect much use of this ladder: namely, this ladder appears to be covered by Hamburg's U.S. patent application 17/916,979. The patent application was assigned to Cryptography Research, a subsidiary of Rambus, and was filed before [124] was published.

9 Twist security

Imagine a careful implementation of NIST P-256 ECDH. The implementation multiplies the user's long-term secret key n by an incoming public key (x, y), using the scalar-multiplication methods from Section 8.1. The implementor is aware of timing attacks (see Section 8.3), and manages to eliminate all timing variations from the software. Also, let's be trendy here: the implementation is using computer-checked proofs that the arithmetic in $\mathbb{F}_p$ is always correct. The software isn't competitive in speed with X25519, but let's assume it's fast enough for the application. Everything is good at this point, right?

Unfortunately not. The implementation never checked that the incoming point (x, y) is on the curve. This has no effect on the normal operation of the protocol—but it exposes the software to an "invalid-curve attack" [54] where an attacker quickly extracts n by sending a few fake points (x, y). See Section 9.2.

We have seen again and again that implementations are vulnerable to invalid-curve attacks. For example:

- Vulnerability announcement CVE-2019-9155 [97] said that the OpenPGP.js software "allows an attacker who is able provide forged messages and gain feedback about whether decryption of these messages succeeded to conduct an invalid curve attack in order to gain the victim's ECDH private key".
- Vulnerability announcement CVE-2021-3798 [201] said that the "openCryptoki Soft token does not check if an EC key is valid when an EC key is created via `C_CreateObject`, nor when `C_DeriveKey` is used

with ECDH public data. This may allow a malicious user to extract the private key by performing an invalid curve attack".

- Vulnerability announcement CVE-2023-46324 [175] said that the free5GC udm software "allows an Invalid Curve Attack because it may compute a shared secret via an uncompressed public key that has not been validated. An attacker can send arbitrary SUCIs to the UDM, which tries to decrypt them via both its private key and the attacker's public key". SUCIs are 5G's "Subscription Concealed Identifiers". See [107] for more information.

Rather than blaming an apparently neverending series of implementors for not following security advice, the SafeCurves criteria have always proactively addressed this attack by moving to better curves:

- First, choose curves meeting the SafeCurves ladder criterion covered in Section 8, so that the incentives explained in that section encourage protocols to send just x rather than (x, y).
- Second, choose curves to be "twist-secure". This section covers the SafeCurves twist-security criteria, after explaining the motivating attacks.

The effectiveness of this approach is illustrated by a direct comparison in the 5G example cited above. According to [107, page 10], 5G requires implementations to support both Curve25519 and P-256 as options for SUCIs. As one would expect from the considerations in Section 8, the protocol is designed to send just x for Curve25519 but (x, y) for P-256. The attack from [107] worked against the P-256 option and failed against the X25519 option in the same software.

9.1 Small-subgroup attacks

Before reviewing invalid-curve attacks, we review **small-subgroup attacks** on ECDH. Small-subgroup attacks for multiplicative groups were introduced in a 1997 paper [165] by Lim and Lee.

As usual, we assume that E is an elliptic curve over $\mathbb{F}_p$, and that $G \in E(\mathbb{F}_p)$ is an ECDH base point of large prime order ℓ. This forces $\#E(\mathbb{F}_p) = h\ell$ for some positive integer h, called the "cofactor". We focus on the case that h is small, as in typical curve choices: e.g., $h = 8$ for Curve25519, and $h = 1$ for NIST P-256.

A small-subgroup attack proceeds as follows. Instead of sending a legitimate curve point eG to Bob, Eve sends Bob a point $Q \in E(\mathbb{F}_p)$ of small order, pretending that Q is her public key. Bob computes nQ as usual, where n is Bob's secret key; computes a hash of nQ as a shared secret key for, e.g., AES-GCM; and uses AES-GCM to encrypt and authenticate data. Because Q has small order, there are not many possibilities for nQ; Eve can simply enumerate the possibilities and check which possibility successfully verifies the data. This attack reveals n modulo the order of Q.

The only possible orders of curve points Q are

- divisors of h and
- ℓ times divisors of h.

In the second case, Q has order at least ℓ, giving too many possibilities for nQ to enumerate, so the attack does not work. Eve's best strategy—assuming the curve is cyclic, which typical curve choices are—is then to take a curve point Q of order h, so that the attack reveals n modulo h.

At this point we distinguish three scenarios. The first scenario is that Bob chose n as a uniform random integer modulo $h\ell$. Then, after this attack, there are still ℓ equally likely possibilities for n.

The second scenario is that Bob chose n as hs, where s is a uniform random integer modulo ℓ. Then the attack reveals nothing, and there are still ℓ equally likely possibilities for n. In both of these scenarios, Eve's best strategy is to continue with the rho method (see Section 4) in the group generated by G, a group of prime order ℓ.

The third scenario is that Bob instead chose n as a uniform random integer modulo ℓ. In this scenario, there is a slight loss of security: the attack reduces n to just ℓ/h possibilities, allowing a "kangaroo" computation [**197**], roughly $\sqrt{h}$ times faster than the rho method.

An implementor can stop a small-subgroup attack by rejecting any Q for which $hQ = 0$, either by carrying out a short computation or by checking against a precomputed list. But this creates a conflict between simplicity and security. An implementation that does not include this check is simpler and more likely to be produced, and will pass typical functionality tests.

A curve designer can protect against this type of attack by choosing curves with $h = 1$. A protocol designer can protect against this type of attack for any curve by specifying $n = hs$. Even without any defenses, the impact is limited to $\sqrt{h}$ for ECDH.

9.2 Invalid-curve attacks

Much more serious is an **invalid-curve** attack. In this case Eve sends Bob a point Q of small order *on another curve*.

For example, instead of sending Bob a point (x, y) satisfying a standard short Weierstrass equation $y^2 = x^3 + ax + b$, Eve sends a point (x, y) satisfying another short Weierstrass equation $y^2 = x^3 + ax + c$ where c is different from b. The standard formulas for scalar multiplication on short Weierstrass curves do not involve the constant coefficient b, so they automatically also work for $y^2 = x^3 + ax + c$. Bob will successfully compute $n(x, y)$ without realizing that anything is amiss.

The advantage of an invalid-curve attack, compared to a small-subgroup attack, is that Eve has many more points Q to choose from. Eve can run the attack using a point Q_2 of order 2 on one curve, a point Q_3 of order 3 on another curve, a point Q_5 of order 5 on another curve, etc., revealing n modulo 2, modulo 3, modulo 5, etc. Soon Eve has enough information to interpolate Bob's entire public key n using an explicit form of the Chinese remainder theorem.

Quantitatively, for fixed non-zero a, the short Weierstrass curves $y^2 = x^3 + ax + b$ cover roughly 25% or roughly 50% of all isomorphism classes of elliptic

curves, depending on p. These curves have roughly $4\sqrt{p}$ different orders, and a huge number of points Q of small order.

Invalid-curve attacks were introduced in a 2000 paper [54] by Biehl, Meyer, and Müller. See also [55] for a variant breaking a Bluetooth security mechanism in which keys are used only once and x-coordinates are authenticated; the attack replaces (x, y) with $(x, 0)$.

Standards typically say that an implementation that receives a point (x, y) from another party *must* check whether (x, y) is on the curve. This takes very little computation time compared to scalar multiplication, and if it is done then it stops an invalid-curve attack. However, this again creates a conflict between simplicity and security. The CVEs show that this is still a frequent problem, more than 20 years after the attack was introduced.

9.3 Point compression

A protocol designer might try to help protect against invalid-curve attacks by specifying point compression. The idea is that an implementation that receives a compressed point, such as x and just one bit of y, will naturally detect invalid inputs when it reconstructs the missing coordinate. On the other hand, if an implementation was not checking the curve equation, then one has to ask what will happen if the implementation does not check squareness as part of a square-root computation to recover y; this question needs analysis. See Section 11.5.

Despite the obvious size advantage and potential security advantage of sending compressed points, it is common for protocols built on short Weierstrass curves to send uncompressed (x, y). We point out three factors that appear to have contributed to this.

The first factor is U.S. patent 6141420, which was filed in 1994 and expired in 2014. Claim 29 of the patent is on communicating a curve point by communicating one coordinate and having the recipient recover the other coordinate. Claim 30 of the patent is more specifically on also sending "identifying information of said other coordinate"—e.g., one bit of y.

Bodo Möller pointed out that point compression had already appeared in a 1992 paper [127, page 171] on ECDH, more than a year before the patent was filed. The patent also shared a coauthor with the paper. We doubt that the patent would have survived litigation. But the patent holder, Certicom, sent many letters regarding its patents, including the point-compression patent; see, e.g., [78]. In 2007, Certicom took Sony to court regarding other patents. It is understandable that protocol designers were hesitant to consider point compression.

(Certicom also filed, in 1997, a patent application on point validation, checking whether points are on curves. Certicom received U.S. patent 7215773 in 2007. That patent has also expired.)

A second factor weighing against point compression is the cost of recovering y, essentially the aforementioned square-root computation in $\mathbb{F}_p$. This involves some extra code and adds roughly 10% to the cost of scalar multiplication.

A third factor is the installed base. Once there are enough protocols and libraries using uncompressed points (x, y) on short Weierstrass curves, there is an incentive for new protocols to do what is most easily handled by the existing libraries, and for new libraries to focus on what is needed by the existing protocols. To be clear, we are not saying that something new can never be deployed; for example, [169] says that the "vast majority" of TLS connections are now using X25519. See also [72] and [73].

9.4 Twist attacks against ladders

Curves meeting the SafeCurves ladder criterion (see Section 8.4) naturally end up with protocols sending just single coordinates as ECDH public keys. This drastically limits the power of invalid-curve attacks.

Consider, for example, a Montgomery curve $By^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$. Any input x that is not on the curve is guaranteed to be on the "twisted" curve $(B/u)y^2 = x^3 + Ax^2 + x$, where u is a non-square in $\mathbb{F}_p$. Specifically:

- If $(x^3 + Ax^2 + x)/B$ is a nonzero square in $\mathbb{F}_p$ then x represents two points $(x, \pm\sqrt{(x^3 + Ax^2 + x)/B})$ on the original curve.
- If $(x^3 + Ax^2 + x)/B$ is a non-square in $\mathbb{F}_p$ then x represents two points $(x, \pm\sqrt{(x^3 + Ax^2 + x)u/B})$ on the twisted curve.
- If $(x^3 + Ax^2 + x)/B$ is zero then x represents one point $(x, 0)$ on each curve.

The Montgomery ladder formulas for $By^2 = x^3 + Ax^2 + x$ also compute scalar multiplication for the twisted curve $(B/u)y^2 = x^3 + Ax^2 + x$, so the attacker can use points of small order on either of these curves, but the single input coordinate does not provide any other attack options. An invalid-curve attack using the twisted curve is called a "twist attack".

The general picture is that single-coordinate ladders work for curves isomorphic to the original curve and for one other isomorphism class of curves, namely all the **nontrivial quadratic twists** of the original curve. If the original curve has $p+1-t$ points then any nontrivial quadratic twist has $p+1+t$ points. Often a nontrivial quadratic twist is called "the twist".

An ECC implementor can stop an invalid-curve attack against ladders by checking whether the input coordinate x belongs to a point Q on the correct curve equation; this requires determining whether $x^3 + Ax^2 + x$ is a square. This computation is doable but noticeable in the overall computation of nQ; this also creates yet another conflict between simplicity and security.

A curve designer can protect against this attack by choosing better curves, namely twist-secure curves, meaning that the twist also has a small cofactor. This renders the checks unnecessary. See Section 9.5 for a quantitative analysis.

Twist-secure curves for DH were proposed in a 2001 talk [21] and posting [22] by one of the authors of this paper. See also the Curve25519 paper [23] and 2008 Fouque–Lercier–Réal–Valette [102].

9.5 Security against twist attacks

SafeCurves requires single-coordinate ladder ECDH to remain secure even if

- Bob chooses n only up to ℓ;
- Bob does not multiply n by the cofactor h for the original curve;
- Bob does not multiply n by the cofactor h' for the twist; and
- Bob does not bother to check whether incoming points are on the original curve.

Specifically, SafeCurves quantitatively evaluates **combined attacks** that use small-subgroup attacks as described above together with invalid-curve attacks using the twist. SafeCurves requires the security level against these attacks to be at least the square root of the lower limit on ℓ in the SafeCurves rho criterion. That lower limit was 2^{200} when we posted the SafeCurves web pages, so SafeCurves required at least 2^{100} security against these attacks, but note that we are now updating this limit; see Section 4.

If both cofactors are very small then the security level of this combined attack is close to the standard rho security level: for example, the combined attack costs $2^{120.3}$ for NIST P-256, and $2^{124.3}$ for Curve25519. In other cases the security level of this combined attack can be far below the standard rho security level: for example, the combined attack costs just $2^{58.4}$ for NIST P-224, and just $2^{44.5}$ for brainpoolP256t1.

Here are two examples showing how to optimize combined attacks:

1. Assume that the original curve has order $h\ell$ and that the twist has order $h'\ell'$ where ℓ and ℓ' are primes around 2^{200}, h and h' are around 2^{50}, and h and h' are coprime. Assume that Bob chooses n as a number less than ℓ. The attacker computes n modulo h in at most 2^{50} operations (trying all h possible values of nQ against some AES-GCM encrypted text); computes n modulo h' in at most 2^{50} operations; obtains n modulo hh' using CRT; and does a kangaroo attack against the $\ell/(hh')$ possibilities of n in time 2^{50}, for a total of $2^{51.6}$ operations. If h or h' factor further, the first two searches can be sped up at the expense of more interaction with Bob; however, this does not reduce the overall running time.
2. Assume instead that ℓ' is around 2^{94}; that ℓ is around 2^{200}; that h' is a product of primes q, r, and s around 2^8, 2^{18}, and 2^{90}; that h is around 2^{10}; and again that h and h' are coprime. Assume again that Bob chooses n as a number less than ℓ. In this situation the best attack is as follows: compute n modulo h in about 2^{10} operations; compute n modulo q and r in about 2^{18} operations; obtain n modulo hqr using CRT; apply a kangaroo attack to the remaining $\ell/(hqr)$ possibilities for n. Here hqr is around 2^{36}, so the kangaroo attack takes only about 2^{82} operations. Note that the attacker did not use a point of order s here, since searching all the multiples of the point would have taken 2^{90} operations; the combined attack balances the cost of the brute-force searches with the cost of a kangaroo attack on the remaining DLP in the main subgroup. Note also that a standard ECDLP problem for

the group of order ℓ' would have been much easier to solve, using only 2^{47} operations, but would have required Bob to expose a twisted curve point nQ to the attacker, rather than using a hash of nQ to encrypt data.

9.6 ECDLP security for the twist

SafeCurves also imposes all of its ECDLP security requirements upon the twist, specifically upon a subgroup of order ℓ', where ℓ' is the largest prime factor of $p+1+t$:

- ℓ' is required to reach the same lower limit as in Section 4.1, to protect against rho attacks. The rho security of this group is labeled **twist rho** in the SafeCurves tables.
- ℓ' is required to be different from p; i.e., the number of points on the original curve must not be $p+2$. This requirement avoids additive transfers (see Section 5) for the twist.
- The embedding degree for ℓ' is required to be at least $(\ell'-1)/100$. This requirement avoids multiplicative transfers (see Section 5) for the twist.

The field discriminant for the twist (see Section 6) is the same as the field discriminant for the original curve, so the twist does not need to be checked separately.

Some of the ECDLP security requirements for the twist are overkill for DH on the original curve: DH does not actually reveal nQ to Eve, so there is no obvious way for Eve to apply (e.g.) an additive transfer. There are, however, other ECC protocols that make full use of both the original curve and its twist, and twist security is important for these protocols. See, e.g., [**148**], [**147**], and [**64**].

10 Completeness

One of the pitfalls of using $E(\mathbb{F}_q)$ instead of $\mathbb{F}_q^*$ is that formulas for the group operations in $E(\mathbb{F}_q)$ do not always work correctly. See Section 10.1 for examples.

When implementations use sometimes-malfunctioning formulas, attackers can trigger the failure cases inside typical ECC protocols (by sending malicious inputs, as in Section 9), and in some cases can learn secret information by analyzing the responses. See, e.g., the 2002 "exceptional procedure attack" [**142**] by Izu and Takagi.

One fix is to mathematically characterize the failure cases and have implementations switch to correct formulas for those cases when those inputs appear. Such switches produce timing variations, which are sometimes exploitable as in Section 8.3; eliminating those timing variations takes further work. Even when the overheads here do not create a problematic slowdown, there is certainly more code. In short, these failure cases are another source of tension between simplicity and security.

One virtue of Montgomery curves is that the Montgomery ladder for ECDH (see Section 8.2) turns out to have no failure cases. To be more precise, given

$X_0(P)$ as input, the Montgomery ladder always produces $X_0(nP)$ as output; here X_0 is a modified x-coordinate, where $X_0(P)$ means 0 if P is the point at infinity and $X_0(P)$ means the x-coordinate of P otherwise. This was proven in [23, Appendix B] for Montgomery curves $y^2 = x^3 + Ax^2 + x$ with a unique point of order 2, i.e., with $A^2 - 4$ not a square, and in our 2017 paper [45, Theorem 4.5] for general Montgomery curves.

But what about protocols beyond ECDH? As an important example, what about elliptic-curve signatures? Elliptic-curve signature verification involves double-scalar multiplication, mapping m, n, P, Q to $mP + nQ$. The literature has various ways to adapt ladders to this situation (see, e.g., [45, Section 4.7]), but so far these adaptations do not have the impressive combination of speed, simplicity, and security provided by the original Montgomery ladder.

Another type of scalar multiplication for which the Montgomery ladder is not the speed leader is single-scalar *fixed-base-point* scalar multiplication. This means, for a fixed base point P, computing nP given n. Fixed base points appear in the key-generation step in ECDH and signatures, and in signature generation. For fixed base points one can precompute many more multiples than the $3P, 5P, \ldots, 15P$ mentioned in Section 8.1, with the result that nP can be computed very efficiently by summing up a secret selection of these precomputed points (for details see, e.g., [31]). This is faster than the Montgomery ladder. For all curve shapes, implementors pursuing speed for fixed-base-point scalar multiplication had an incentive to fall back to incomplete addition formulas.

This section explains how the underlying problem was resolved, starting with the advent of Edwards curves in 2007—which, fortunately, turn out to be compatible with Montgomery curves such as Curve25519. Before looking at the details, we point out one way to see how successful this has been.

ECDSA [8] is a standard signature system using short Weierstrass curves. EdDSA [31] is a now-standard signature system using Edwards curves. There is a long history of timing attacks against implementations of ECDSA, such as the following:

- The 2019 "TPM-FAIL" paper [176] recovered ECDSA secret keys from Trusted Platform Modules that had been manufactured by STM and Intel. Both TPMs were certified under FIPS 140-2. The STM TPM was also Common Criteria certified at EAL4+.
- The 2019 "Minerva" paper [144] recovered ECDSA keys from a FIPS-certified CC-certified Athena IDProtect smart card, pointed out seven other certified devices using the same ECDSA implementation, and reported timings indicating that a similar attack would work against ECDSA implementations in 4 out of 13 software libraries.
- In December 2023, Mozilla [178] announced that, in the Firefox browser, "multiple NSS NIST curves were susceptible to a side-channel attack known as 'Minerva'."

The case of Minerva allows a direct comparison of how well ECDSA and EdDSA held up against these attacks. Out of the 13 software libraries covered in [144], 9 already supported EdDSA. In particular, out of the 4 software libraries where

the timings in [144] indicated that ECDSA would be exploitable (libgcrypt, MatrixSSL, JDK, and Crypto++), all except JDK already supported EdDSA. None of the 9 EdDSA implementations were vulnerable.

The analysis in [25] shows that, in 8 of 9 cases, the underlying EdDSA code was designed to be constant-time—something that is easier to achieve for EdDSA than for ECDSA. This is connected to the choice of curves: Edwards curves make constant-time software easier than short Weierstrass curves do, for reasons explained below. The other case, libgcrypt, had variable-time EdDSA code but was rescued by a curve-independent feature of EdDSA, illustrating Section 1's comment about the word "most".

10.1 Incompleteness

We now consider again the P1363 algorithm from Section 8.1, computing nP given an integer n and a point P on a short Weierstrass curve. In applications where a further slowdown is acceptable, an implementor might eliminate the subtractions and use a simpler inner loop that looks like this:

- $Q \leftarrow Q + Q$.
- $Q \leftarrow Q + P$ if the current bit of n is set.

The simplest way to implement $+$ is to copy a readily available addition formula, such as one of the formulas cited in Section 8.1. The implementor then finds that this does not work: the scalar-multiplication formulas consistently fail random tests. The problem is that the so-called "addition formula" does not always work: in particular, it fails for the doublings $Q + Q$.

Because this implementation fails random tests, it will be fixed. The simplest fix has an inner loop that looks like this, with a "doubling formula" plugged in for $2Q$:

- $Q \leftarrow 2Q$.
- $Q \leftarrow Q + P$ if the current bit of n is set.

This passes random tests. Unfortunately, it still fails if Q happens to match P. This will *not* be caught by random tests.

A different fix is to modify "+" to check for its inputs being equal. But this produces a slower and more complicated implementation—and still does not catch all the failure cases. For example, the standard Weierstrass addition formula fails if Q happens to match $-P$. This is something else that will not be caught by random tests.

A typical presentation of the complete group operation on a short Weierstrass curve involves separate formulas for six different cases. This number of cases is not optimal: a 1995 theorem from Bosma and Lenstra [63, Theorem 1] states that "The smallest cardinality of a complete system of addition laws on E equals two". One might think that this means that any *single* addition formula must have failure cases; but, as we will see in a moment, the facts are more subtle than that.

10.2 Edwards curves

The original Edwards paper [95] stated that "The normal form $x^2 + y^2 = a^2 + a^2x^2y^2$ for elliptic curves simplifies formulas in the theory of elliptic curves and functions" and presented a remarkably simple addition formula for this curve shape. The constant a is required to be nonzero and to have $a^4 \neq 1$.

Replacing x and y with ax and ay respectively, and dividing by a^2, gives the curve shape $x^2 + y^2 = 1 + a^4x^2y^2$. In [36], we suggested generalizing this to $x^2 + y^2 = 1 + dx^2y^2$ for any $d \notin \{0, 1\}$, and showed that this includes a curve birationally equivalent over $\mathbb{F}_p$ to Curve25519. In this level of generality, the Edwards addition formula says that

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

We showed that addition in projective coordinates takes only 11**M** plus a multiplication by d, and that doubling takes only 7**M**. We also showed that, for Edwards curves where d is *not* a square, the Edwards addition formula is complete—it adds every pair of points correctly.

Why does this completeness not contradict the Bosma–Lenstra theorem? A superficial answer is that the theorem is stated only for Weierstrass curves; but one can see that the same concept applies to any shape of elliptic curve. The real answer is that the Bosma–Lenstra definition of incompleteness does not force failure cases for $E(\mathbb{F}_p)$; it forces failure cases for $E(K)$ for some field K *containing* $\mathbb{F}_p$. This is consistent with our completeness theorem—it simply means that d must be a square in K. Those failures do not affect computations in the group $E(\mathbb{F}_p)$ when d is not a square in $\mathbb{F}_p$.

Followup work found even better speeds for arithmetic on Edwards curves, and, more generally, for "twisted Edwards curves" $ax^2 + y^2 = 1 + dx^2y^2$, which were introduced in [27]. Any Montgomery curve is birationally equivalent to a twisted Edwards curve and vice versa. The speed records for addition are just 8**M** plus a multiplication by d, using complete formulas introduced by Hisil, Wong, Carter, and Dawson in a 2008 paper [130, Section 3.1] for the case $a = -1$ (which is compatible with, e.g., Curve25519). These formulas rely on "extended coordinates" introduced in the same paper; doublings take 8**M** in those coordinates, but skipping the extra coordinate when it is not needed produces doublings in 7**M** and additions in 8**M**, as explained in [130, Section 4.3].

The same paper [130] also showed that for addition one could achieve 8**M** *without* a multiplication by d, with the caveat of those formulas not being complete. There is a tension here between speed and security, but fortunately a very small tension for curves chosen to have small d.

In 2011, together with Duif, Schwabe, and Yang, we introduced the EdDSA signature system [31] (see also the extended version in [32]), and specifically Ed25519, which is EdDSA using Curve25519. As mentioned above, EdDSA relies on Edwards curves. The speed, simplicity, and completeness of Edwards addition contribute directly to the speed, simplicity, and security of Ed25519 software.

10.3 Completeness for more curves

Back in 2008, in joint work with Farashahi [47], we introduced "binary Edwards curves" as a new curve shape and designed complete addition formulas for it. In 2009, in joint work with Kohel [35] (see also [30]), we introduced a complete addition formula for "twisted Hessian curves". Finally, also in 2009, we introduced a complete addition formula that can safely be conjectured to cover all elliptic curves over non-binary finite fields; see [24]. We checked that all of the non-binary NIST curves were covered; see [24, page 23] for the case of NIST P-256. Arene, Kohel, and Ritzenthaler showed in 2012 [14, Theorem 4.3] that every elliptic curve over a large finite field has a complete addition formula in Weierstrass form.

Saying that a curve has a complete addition formula does not imply that sensible implementors will want to use the formula. Sometimes a complete addition formula is simple and fast, as the Edwards addition formula illustrates; but sometimes a complete addition formula is slower and more complicated than constant-time software that correctly merges two or more incomplete formulas.

In 2013, the SafeCurves web pages reviewed the basic completeness results for the Montgomery ladder and the Edwards addition formula, and continued as follows:

> Subsequent research has introduced other complete scalar-multiplication formulas. However, many of these formulas are considerably slower and more complicated than standard incomplete scalar-multiplication formulas, creating major conflicts between simplicity, efficiency, and security.
>
> SafeCurves requires curves to support simple, fast, *complete*, constant-time single-coordinate single-scalar multiplication. This includes the SafeCurves ladder requirement but goes further by requiring completeness. SafeCurves also requires curves to support simple, fast, *complete*, constant-time multi-scalar multiplication.

See Section 8 for the definitions of "simple" and "fast".

This SafeCurves criterion is satisfied by, e.g., Curve25519: the Montgomery ladder provides simple, fast, complete, constant-time single-coordinate single-scalar multiplication, and standard multi-scalar-multiplication techniques on top of the Edwards addition formula provide simple, fast, complete, constant-time multi-scalar multiplication. Note that "simple" and "fast" are relative to single-scalar multiplication in the first case and relative to multi-scalar multiplication in the second case; fast multi-scalar multiplication on top of the Edwards addition formula is certainly not as simple as the Montgomery ladder.

There has been further work on completeness for other curve shapes. In 2015, Renes, Costello, and Batina [202] showed that one of the Bosma–Lenstra formulas takes just 12**M** for short Weierstrass curves E in projective coordinates; this particular formula is complete for $E(\mathbb{F}_p)$ if $\#E(\mathbb{F}_p)$ is odd. However, the complete doubling formulas in [202] for the same coordinate system take 11**M**. This approach to complete scalar multiplication is much slower than incomplete

scalar multiplication on the same curves (see [211] for quantification), never mind questions of simplicity; so it does not meet this SafeCurves criterion.

There is a statement in [202] that the formulas in that paper are "comparably efficient". That statement appears to be a comparison to earlier *complete* formulas for the same curves. What matters for this SafeCurves criterion is instead the comparison to the simplest, fastest scalar-multiplication algorithms for the same curves *without regard to security*. This is also what matters for predicting security risks.

Vulnerability announcement CVE-2023-24532 [120] illustrates how powerful the implementation incentives are. There was a valiant effort to deploy the formulas from [202] as part of the "crypto/elliptic" library for the Go programming language—but the library ended up using faster P-256 software instead. The vulnerability announcement indicates that the faster software has a bug stemming from its use of incomplete formulas: "The ScalarMult and ScalarBaseMult methods of the P256 Curve may return an incorrect result if called with some specific unreduced scalars (a scalar larger than the order of the curve)." It is not clear whether the library has applications where the bug is exploitable. What is clear is that larger tensions between simplicity, efficiency, and security make it more likely that security will be lost.

11 Strings as group elements

Beyond the core ECC tasks of ECDH as in X25519 and signatures as in Ed25519, there is a vast literature on further cryptographic protocols using elliptic curves. Complications arise in many of these protocols because of the interface gap between

- bit strings—the standard interchange format for data stored inside computers—and
- elliptic-curve points—the central objects appearing in ECC.

This section introduces the central issue, and then explains a simple workaround known for *some* curves.

11.1 The gap between strings and curve points

In symmetric cryptography, stream encryption of a message m produces ciphertext $c = m \oplus s$. Here s is the output of, e.g., AES-CTR applied to a "nonce" (a number used once; e.g., number i for the ith message) and a secret key; and $\oplus$ is the exclusive-or operation on bit strings.

For comparison, consider the ElGamal encryption system using the multiplicative group $\mathbb{F}_p^*$. Alice's secret key is a, and Alice's public key is g^a for a fixed generator $g \in \mathbb{F}_p^*$. To send a message m, Bob generates n randomly for this message, and sends the pair (g^n, mg^{an}). Alice divides mg^{an} by g^{an} to reconstruct m.

In this ElGamal example, m is implicitly an element of $\mathbb{F}_p^*$, whereas one expects a message to instead be a bit string. This distinction might not seem worth commenting on:

- There is a standard representation of elements of $\mathbb{F}_p^*$ as strings of $\lceil \log_2 p \rceil$ bits, namely the usual little-endian encodings of the integers $1, 2, \ldots, p-1$.
- If m is a message consisting of, say, $\lfloor \log_2(p-1) \rfloor$ bits then one can zero-pad m to $\lceil \log_2 p \rceil$ bits, replace the all-zero string with the encoding of $p-1$, and observe that the result is the representation of some element of $\mathbb{F}_p^*$.

In short, there is an efficient bijection between strings of $\lfloor \log_2(p-1) \rfloor$ bits and a large subset of the group elements. This provides a clean interface for protocols viewing strings as group elements.

The semantic gap here becomes more of a problem in the elliptic-curve version of the ElGamal system. Alice's public key is then aG for a fixed base point $G \in E(\mathbb{F}_p)$, and Bob encrypts $m \in E(\mathbb{F}_p)$ as the pair $(nG, m + naG)$. How does one view bit strings as elements of $E(\mathbb{F}_p)$?

There are again standard representations of elements of $E(\mathbb{F}_p)$ as strings. Consider, for example, a compressed representation of elements of $E(\mathbb{F}_p)$ as in Section 9.3 when E is a short Weierstrass curve: a point (x, y) is represented as x and one bit of y (and the point at infinity is represented as some otherwise unused string; this exists by Hasse's theorem when p is large), taking $\lceil \log_2 p \rceil + 1$ bits. The problem is that one cannot reliably obtain strings in this representation by zero-padding slightly shorter messages: $x(E(\mathbb{F}_p))$ covers only about half of the elements of $\mathbb{F}_p$, and not simply an interval as in the $\mathbb{F}_p^*$ case. So this representation does not seem useful for a protocol that needs to reliably encode bit strings as curve points.

11.2 Protocols using strings as group elements

One can dismiss the ElGamal example from Section 11.1 by saying that its basic purpose, namely encryption, is better achieved by combining DH with a symmetric cipher, as proposed in the original DH paper. When Section 9.1 mentioned producing, e.g., an AES-GCM key by hashing an ECDH shared secret, it was implicitly representing the ECDH shared secret as a bit string for input to the hash function. This relies on mapping elliptic-curve points *to* bit strings; it does not need a bijection. Similarly, EdDSA does not need a bijection.

However, more advanced protocols frequently encounter the problem of viewing strings as group elements. Here is an illustrative example, namely password-authenticated key exchange (PAKE).

The goal of PAKE is to upgrade a not-very-high-entropy password shared by Alice and Bob into a high-entropy shared secret. In the simplest PAKE protocol, Alice generates an integer m and uses (a hash of) the password as a key for a block cipher to encrypt the block mG; Bob generates an integer n and similarly encrypts nG; Alice and Bob each compute (a hash of) mnG. For each subsequent communication session, Alice and Bob repeat this process with fresh integers m

and n. (One might wonder why Alice and Bob do not reuse secrets across sessions, or share a higher-entropy secret in the first place; the typical answer is that Alice and Bob are humans with limited memory and want their devices to forget all secrets as quickly as possible.)

An attacker starting with a correct guess of the password before the protocol trivially breaks security of this protocol by forging messages, but this has probability only $1/W$ of success per session if there are W equally likely passwords, so on average it takes about $W/2$ sessions before success.

A much more efficient "partition attack", introduced by Patel [191] in 1997 for multiplicative groups and adapted to elliptic-curve groups in 2001 by Boyd, Montague, and Nguyen [64, Section 5.1], is as follows: try decrypting the encrypted mG and nG using various guesses for the password, and reject any guess for which the results are not group elements. The critical observation here is that most block-length strings are not encodings of group elements. Consequently, these trials rapidly reject most passwords by observing a single session, and confidently identify the correct password after a logarithmic number of sessions—at which point an active attack breaks security with probability 1. Even if point mG is represented by its x-coordinate $x(mG)$, only about half of all strings give valid x-coordinates, still permitting a partition attack to succeed in about $\log_2 W$ steps.

For multiplicative groups $\mathbb{F}_p^*$, it is easy to block partition attacks by equating block-length strings with group elements as in Section 11.1 and taking g to generate $\mathbb{F}_p^*$ (rather than to have prime order). This does not equate strings with *all* group elements—if g^m does not correspond to a string then Alice has to try again; same for Bob—but all block-length strings are covered by the elements g^m that correspond to strings. For p of the form $p = 2^t - s$ for very small s it is also safe to take t-bit strings, which mean no retries, as the chance that any candidate decryption ever lands in the forbidden interval $[p, 2^t - 1]$ is negligible.

For elliptic-curve groups $E(\mathbb{F}_p)$, a more complicated protocol "secure against partition attacks" was proposed in [64, Section 5.2]. Alice encrypts either $x(mG)$ or $x(m'G')$, chosen randomly for each session, where G generates $E(\mathbb{F}_p)$ and G' generates the twist. The idea here is that these x-coordinates cover $\mathbb{F}_p$, and thus cover all strings (after retries as in the previous paragraph). Bob sends back unencrypted points $x(nG)$ and $x(n'G')$, and then Alice and Bob use either $x(mnG)$ or $x(m'n'G')$, depending on whether Alice had chosen $x(mG)$ or $x(m'G')$ in the first place.

The literature has many more examples of complications stemming from the interface gap between strings and elements of $E(\mathbb{F}_p)$; see, e.g., the references in our 2013 paper [34] with Hamburg and Krasnova. As an illustration of these complications being a security risk, [34] pointed out the following active attack against the protocol from [64]: the attacker replaces $x(n'G')$ with random data, and, if the protocol continues successfully, concludes that Alice was using $x(mG)$ in the first place. A partition attack then eliminates half of the passwords, and repeating for a logarithmic number of sessions breaks the protocol.

11.3 Strings as curve points: a dangerous approach

A simple-sounding, and presumably correct, method to view bit strings as curve points is as follows. Take bit strings that are, say, 10 bits shorter than compressed point representations. Given a bit string m, consider the 1024 possible point representations starting with m, and take the lexicographically smallest that represents a point. This fails if none of the 1024 strings represents a point; presumably such failures do not occur for, e.g., 256-bit primes p.

One issue with this (presumed) bijection is that stopping after the first point representation that works—as part of mapping a bit string to a point, or as part of deciding whether a point corresponds to a string—leaks information through timing. Constant-time software instead has to try all 1024 possibilities. Another issue is that the bijection does not cover a large fraction of curve points. Protocols trying group elements until they find elements represented by strings, as in some of the protocols considered in Section 11.2, might have to try thousands of times.

In 2019, Vanhoef and Ronen [**221**] announced Dragonblood, breaking every analyzed implementation of the WPA3 Dragonfly handshake. Most implementations were vulnerable to invalid-curve attacks as in Section 9.2, but there were exceptions, such as hostapd, which [**221**] instead broke with a timing attack. Dragonfly used a lexicographic computation of a point from a bit string, and the timing attack targeted this computation.

Another implementation covered in [**221**], namely FreeRADIUS, stopped the lexicographic map after just 10 possibilities. This was not a problem for functionality: >99.9% of handshakes would succeed, and users would not notice failure cases if handshakes were retried automatically. Security was a different story: the failure cases were efficiently exploitable, as shown in [**221**, Section 7]. FreeRADIUS was also vulnerable to invalid-curve attacks.

11.4 Strings as curve points: a better approach

A 2006 paper by Shallue and van de Woestijne [**216**] introduced a formula mapping $\mathbb{F}_p$ to a reasonably large fraction of $E(\mathbb{F}_p)$. There are many followup papers refining this "hash-to-curve" idea; see, e.g., the references in [**34**, Section 1.4]. For protocols that simply want to map bit strings to points, such as Dragonfly, these formulas are much nicer for implementations than the lexicographic approach described in Section 11.3. However, "hash-to-curve" formulas are typically not bijections, so they do not address the foundational interface gap highlighted in Section 11.1 and exploited in Section 11.2.

Efficient bijections are known for *some* elliptic curves. For example, for $p \equiv 2 \pmod 3$, the function $y \mapsto ((y^2 - b)^{1/3}, y)$ is an efficient bijection between $\mathbb{F}_p$ and $\{(x, y) : (x, y) \in E(\mathbb{F}_p)\}$, where E is the curve $y^2 = x^3 + b$. This curve has, however, more basic security problems. Miller commented in 1986 [**174**, page 425] that another curve having $p + 1$ points is convenient for calculations but that "it may be prudent to avoid curves with complex multiplication because the extra structure of these curves might somehow be used to give a better algorithm". This avoidance is the topic of Section 6, and these curves with $p+1$

points were then discovered to be weak, specifically because of multiplicative transfers; see Section 5.

More interestingly, efficient bijections are known for essentially all curves E for which $\#E(\mathbb{F}_p) \equiv 0 \pmod 2$, via the "Elligator 2" construction from [34]. For many of the curves, [103] had already introduced a slightly more complex injective map to the curve, which [34] showed how to invert; see also [88] for further techniques and references. For each of these curves E, the fraction of $E(\mathbb{F}_p)$ covered by strings is between about $1/4$ and about $1/2$, so retries as in Section 11.2 are not very expensive. Of course, it would be even better to have an efficient bijection covering all of $E(\mathbb{F}_p)$.

The issues addressed by these bijections are not as central to ECC as the issues from Sections 8, 9, and 10. However, looking at the broader ECC literature shows frequent error-prone contortions to work around the interface gap between strings and points, as illustrated by the PAKE example in Section 11.2 and many more examples cited in [34]. We therefore include an "indistinguishability" criterion in SafeCurves, asking for an efficient constant-time bijection between all b-bit strings and a large fraction of curve points. We allow an undetectable fraction of b-bit strings to be skipped, although one can tweak the known constructions of bijections to avoid this at the expense of replacing b with $b-1$.

11.5 Encodings enforcing group membership

An efficient bijection between the set of b-bit strings and (many) curve points has another interesting property: if a protocol requires a curve point to be communicated as a b-bit string, then decoding always produces a point on the correct curve, automatically avoiding invalid-curve attacks.

This does not mean that a protocol as simple as ECDH should use these encodings: ECDH software is simpler if one uses x-coordinates with the Montgomery ladder, as in Figure 8.2.1, and chooses a twist-secure curve. However, looking at a much wider range of ECC protocols shows that security analyses typically assume that incoming points are in $E(\mathbb{F}_p)$. Rather than trusting each implementation to check this, one can have each protocol use a general-purpose encoding that forces inputs to be in $E(\mathbb{F}_p)$.

Two complications appear at this level of generality. First, one cannot expect all protocols to be willing and able to handle the possibility of a point not being representable as a string. Second, security analyses sometimes assume that incoming points are specifically in the order-ℓ group generated by $G \in E(\mathbb{F}_p)$. This group matches $E(\mathbb{F}_p)$ if $E(\mathbb{F}_p) = \ell$, but the known efficient bijections require larger cofactors.

As an illustration of the second complication, the Monero blockchain announced in 2017 [166] that it had patched a vulnerability allowing each coin to be spent 8 times. Monero used Curve25519, which has cofactor 8, so there are 8 points $T \in E(\mathbb{F}_p)$ such that $8T$ is the neutral element; Monero's security analysis was expecting a point P to be in the order-ℓ group, but the software was accepting $P + T$ as a separate expenditure for each of the 8 points T. Note that switching to cofactor 1 exacerbates the core problem here: it leads

to invalid-curve attacks even against basic ECDH, as illustrated by the recent attacks cited in Section 9.

These considerations have triggered interest in ways to encode elements of the order-ℓ group as b-bit strings, with decoders rejecting all other b-bit strings. What would be best is a bijection, so that nothing needs to be rejected. In the absence of a bijection, some possibilities are the following:

- When $E(\mathbb{F}_p)$ is, e.g., a group of order ℓ in short Weierstrass form, one can encode a curve point as (x, y) (with some handling of the point at infinity, although some protocols will want to reject this point). The decoder is then required to check whether (x, y) is on the curve. To limit the damage just in case the check is omitted, it seems safer to replace y with one bit of y as noted in Section 9.3, using a square-root computation to recover y. However, the strength of this protection is unclear: the literature does not show what will happen if an invalid x is sent and the resulting y is not tested against the putative y^2.
- To handle general curves, one could similarly send (x, y) or a compressed version of (x, y), and add an explicit test whether (x, y) has order ℓ, which seems to be a useful protection in any case. For typical Edwards curves, there are alternative representations of the order-ℓ group that advertise efficient encoding and decoding; see, e.g., [**123**]. With any of these encodings, one has to ask what will happen if the decoder's tests are omitted.

Without a thorough analysis of failure cases, there seems to be no alternative to making sure that the decoding (and encoding) software works correctly, but at least this software can be shared across many protocols asking for an order-ℓ group. Within *correct* implementations of order-ℓ groups, the most efficient groups known rely on Edwards curves, and on the formulas from [**130**].

References

[1] — (no editor), *1997 IEEE symposium on security and privacy, May 4–7, 1997, Oakland, CA, USA*, IEEE Computer Society, 1997. ISBN 0-8186-7828-3. URL: https://ieeexplore.ieee.org/xpl/conhome/4693/proceeding. See [191].

[2] — (no editor), *27th IEEE symposium on computer arithmetic, ARITH 2020, Portland, OR, USA, June 7–10, 2020*, IEEE, 2020. ISBN 978-1-7281-7120-3. URL: https://ieeexplore.ieee.org/xpl/conhome/9146973/proceeding. See [56].

[3] — (no editor), *2020 IEEE symposium on security and privacy, SP 2020, San Francisco, CA, USA, May 18–21, 2020*, IEEE, 2020. URL: https://ieeexplore.ieee.org/xpl/conhome/9144328/proceeding. See [221].

[4] — (no editor), *30th IEEE annual international symposium on field-programmable custom computing machines, FCCM 2022, New York City, NY, USA, May 15–18, 2022*, IEEE, 2022. ISBN 978-1-6654-8332-2. DOI: 10.1109/FCCM53951.2022. See [230].

[5] — (no editor), *IEEE European symposium on security and privacy, EuroS&P 2023—workshops, Delft, Netherlands, July 3–7, 2023*, IEEE, 2023. ISBN 979-8-3503-2720-5. DOI: 10.1109/EuroSPW59978.2023. See [169].

[6] "Bushing", Hector Martin "marcan" Cantero, Segher Boessenkool, Sven Peter, *PS3 epic fail* (2010). URL: https://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf. Citations in this document: §A.

[7] Joe Abernathy, *The following is the written response to my request for an interview with the NSA* (1992). URL: https://archive.epic.org/crypto/dss/nsa_abernathy_letter.html. Citations in this document: §1.4.

[8] Accredited Standards Committee X9, *ANSI x9.62-1999: Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)* (1999). Citations in this document: §1.1, §5, §7.1, §10.

[9] Accredited Standards Committee X9, *ANSI X9.63-2001: Public key cryptography for the financial services industry: key agreement and key transport using elliptic curve cryptography* (2001). URL: https://web.archive.org/web/20010724160111/https://grouper.ieee.org/groups/1363/Research/Other.html. Citations in this document: §1.1.

[10] Onur Aciiçmez, Billy Bob Brumley, Philipp Grabher, *New results on instruction cache attacks*, in CHES 2010 [**168**] (2010), 110–124. URL: https://iacr.org/archive/ches2010/62250105/62250105.pdf. Citations in this document: §8.3.

[11] Alejandro Cabrera Aldaya, Billy Bob Brumley, *When one vulnerable primitive turns viral: novel single-trace attacks on ECDSA and RSA*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020** (2020), 196–221. URL: https://eprint.iacr.org/2020/055. DOI: 10.13154/tches.v2020.i2.196-221. Citations in this document: §A.

[12] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop van de Pol, Yuval Yarom, *Amplifying side channels through performance degradation*, in ACSAC 2016 [**210**] (2016), 422–435. URL: https://eprint.iacr.org/2015/1141. Citations in this document: §A.

[13] Diego F. Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, Yuval Yarom, *LadderLeak: breaking ECDSA with less than one bit of nonce leakage*, in CCS 2020 [**164**] (2020), 225–242. URL: https://eprint.iacr.org/2020/615. DOI: 10.1145/3372297.3417268. Citations in this document: §A.

[14] Christophe Arene, David R. Kohel, Christophe Ritzenthaler, *Complete addition laws on abelian varieties*, LMS J. Comput. Math. **15** (2012), 308–316. DOI: 10.1112/s1461157012001027. Citations in this document: §10.3.

[15] Vijay Atluri, Claudia Díaz (editors), *Computer security—ESORICS 2011—16th European symposium on research in computer security, Leuven, Belgium, September 12–14, 2011, proceedings*, Lecture Notes in Computer Science, 6879, Springer, 2011. ISBN 978-3-642-23821-5. See [75].

[16] Daniel V. Bailey, Brian Baldwin, Lejla Batina, Daniel J. Bernstein, Peter Birkner, Joppe W. Bos, Gauthier van Damme, Giacomo de Meulenaer, Junfeng Fan, Tim Güneysu, Frank Gurkaynak, Thorsten Kleinjung, Tanja Lange, Nele Mentens, Christof Paar, Francesco Regazzoni, Peter Schwabe, Leif Uhsadel, *Breaking ECC2K-130* (2009). URL: https://eprint.iacr.org/2009/466. Citations in this document: §B.

[17] R. Balasubramanian, Neal Koblitz, *The improbability that an elliptic curve has subexponential discrete log problem under the Menezes–Okamoto–Vanstone algorithm*, J. Cryptol. **11** (1998), 141–145. Citations in this document: §5.

[18] James Bamford, *The NSA is building the country's biggest spy center (watch what you say)* (2012). URL: https://www.wired.com/2012/03/ff-nsadatacenter/. Citations in this document: §D.3.

[19] Mihir Bellare (editor), *Advances in cryptology—CRYPTO 2000, 20th annual international cryptology conference, Santa Barbara, California, USA, August 20–24, 2000, proceedings*, Lecture Notes in Computer Science, 1880, Springer, 2000. ISBN 3-540-67907-3. See [54].

[20] Andreas Bender, Guy Castagnoli, *On the implementation of elliptic curve cryptosystems*, in [**68**] (1990), 186–192. MR 91d:11154. Citations in this document: §2.2.

[21] Daniel J. Bernstein, *A software implementation of NIST P-224* (29 Oct 2001). URL: https://cr.yp.to/talks.html#2001.10.29. Citations in this document: §9.4.

[22] Daniel J. Bernstein, *Re: Current consensus on ECC* (1 Nov 2001). URL: https://groups.google.com/g/sci.crypt/c/mu_paShEU3w/m/m491pYxHbtAJ. Citations in this document: §9.4.

[23] Daniel J. Bernstein, *Curve25519: new Diffie-Hellman speed records*, in PKC 2006 [**231**] (2006), 207–228. URL: https://cr.yp.to/papers.html#curve25519. Citations in this document: §1.3, §2.1, §8.2, §8.2, §8.5, §9.4, §10, §B, §E.2.

[24] Daniel J. Bernstein, *Complete addition laws for all elliptic curves over finite fields* (2009). URL: https://cr.yp.to/talks.html#2009.07.17. Citations in this document: §10.3, §10.3.

[25] Daniel J. Bernstein, *Why EdDSA held up better than ECDSA against Minerva* (2019). URL: https://blog.cr.yp.to/20191024-eddsa.html. Citations in this document: §10.

[26] Daniel J. Bernstein, *Cryptographic competitions*, Journal of Cryptology **37** (2024), article 7. URL: https://cr.yp.to/papers.html#competitions. Citations in this document: §7.

[27] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, Christiane Peters, *Twisted Edwards curves*, in Africacrypt 2008 [**223**] (2008), 389–405. Citations in this document: §8.6, §10.2.

[28] Daniel J. Bernstein, Tung Chou, *CryptAttackTester: formalizing attack analyses*, in Crypto 2024, to appear (2023). URL: https://eprint.iacr.org/2023/940. Citations in this document: §B.1.

[29] Daniel J. Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, Christine van Vredendaal, *How to manipulate curve standards: A white paper for the black hat*, in [**79**] (2015), 109–139. URL: https://bada55.cr.yp.to. Citations in this document: §7.2, §7.3.

[30] Daniel J. Bernstein, Chitchanok Chuengsatiansup, David Kohel, Tanja Lange, *Twisted Hessian curves*, in Latincrypt 2015 [**161**] (2015), 269–294. URL: https://eprint.iacr.org/2015/781. DOI: 10.1007/978-3-319-22174-8_15. Citations in this document: §10.3.

[31] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, in CHES 2011 [**199**] (2011), 124–142; see also newer version [**31**]. Citations in this document: §10, §10, §10.2.

[32] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, Bo-Yin Yang, *High-speed high-security signatures*, Journal of Cryptographic Engineering **2** (2012), 77–89; see also older version [**31**]. URL: https://eprint.iacr.org/2011/368. Citations in this document: §10.2, §E.2.

[33] Daniel J. Bernstein, Susanne Engels, Tanja Lange, Ruben Niederhagen, Christof Paar, Peter Schwabe, Ralf Zimmermann, *Faster elliptic-curve discrete logarithms on FPGAs* (2016). URL: https://eprint.iacr.org/2016/382. Citations in this document: §B.3.

[34] Daniel J. Bernstein, Mike Hamburg, Anna Krasnova, Tanja Lange, *Elligator: elliptic-curve points indistinguishable from uniform random strings*, in CCS 2013 [**207**] (2013), 967–980. URL: https://eprint.iacr.org/2013/325. Citations in this document: §11.2, §11.2, §11.4, §11.4, §11.4, §11.4.

[35] Daniel J. Bernstein, David Kohel, Tanja Lange, *Projective coordinates for twisted Hessian curves* (2009). URL: https://hyperelliptic.org/EFD/g1p/auto-twistedhessian-projective.html. Citations in this document: §10.3.

[36] Daniel J. Bernstein, Tanja Lange, *Faster addition and doubling on elliptic curves*, in Asiacrypt 2007 [**157**] (2007), 29–50. URL: https://eprint.iacr.org/2007/286. Citations in this document: §3, §10.2.

[37] Daniel J. Bernstein, Tanja Lange, *Explicit-Formulas Database* (2007), first posted in 2007, also updated afterwards. URL: https://hyperelliptic.org/EFD/. Citations in this document: §8.1.

[38] Daniel J. Bernstein, Tanja Lange, *Analysis and optimization of elliptic-curve single-scalar multiplication*, Contemporary Mathematics **46** (2008), 1–20. URL: https://eprint.iacr.org/2007/455. Citations in this document: §8.1.

[39] Daniel J. Bernstein, Tanja Lange, *Type-II optimal polynomial bases*, in WAIFI 2010 [**128**] (2010), 41–61. URL: https://eprint.iacr.org/2010/069. Citations in this document: §B.

[40] Daniel J. Bernstein, Tanja Lange, *Computing small discrete logarithms faster*, in INDOCRYPT 2012 [**110**] (2012), 317–338. URL: https://eprint.iacr.org/2012/458. Citations in this document: §4.2.

[41] Daniel J. Bernstein, Tanja Lange, *Two grumpy giants and a baby*, in ANTS 2012 [**132**] (2013), 87–111. URL: https://eprint.iacr.org/2012/294. Citations in this document: §B.1.

[42] Daniel J. Bernstein, Tanja Lange, *Security dangers of the NIST curves* (31 May 2013). URL: https://cr.yp.to/talks.html#2013.05.31. Citations in this document: §1.4, §7.2.

[43] Daniel J. Bernstein, Tanja Lange, *SafeCurves: choosing safe curves for elliptic-curve cryptography* (2013). URL: https://safecurves.cr.yp.to. Citations in this document: §1.2, §1.3, §1.4, §7.2, §7.6, §D.1.

[44] Daniel J. Bernstein, Tanja Lange, *Failures in NIST's ECC standards* (2016). URL: https://cr.yp.to/papers.html#nistecc. Citations in this document: §1.4.

[45] Daniel J. Bernstein, Tanja Lange, *Montgomery curves and the Montgomery ladder*, in [**61**] (2017), 82–115. Citations in this document: §8.4, §10, §10.

[46] Daniel J. Bernstein, Tanja Lange, *Failures in NIST's ECC standards, part 2* (2020). URL: https://csrc.nist.gov/files/pubs/sp/800/186/final/docs/sp800-186-draft-comments-received.pdf. Citations in this document: §1.4.

[47] Daniel J. Bernstein, Tanja Lange, Reza Rezaeian Farashahi, *Binary Edwards curves*, in CHES 2008 [**190**] (2008), 244–265. URL: https://cr.yp.to/papers.html#edwards2. Citations in this document: §10.3.

[48] Daniel J. Bernstein, Tanja Lange, Chloe Martindale, Lorenz Panny, *Quantum circuits for the CSIDH: optimizing quantum evaluation of isogenies*, in Eurocrypt 2019 [**140**] (2019), 409–441. URL: https://cr.yp.to/papers.html#qisog. DOI: 10.1007/978-3-030-17656-3_15. Citations in this document: §B, §B, §B.3, §D.1.

[49] Daniel J. Bernstein, Tanja Lange, Ruben Niederhagen, *Dual EC: A standardized back door*, in [**206**] (2016), 256–281. URL: https://www.projectbullrun.org/dual-ec/index.html. Citations in this document: §7, §A.

[50] Daniel J. Bernstein, Tanja Lange, Peter Schwabe, *On the correct use of the negation map in the Pollard rho method*, in PKC 2011 [**77**] (2011), 128–146. URL: https://eprint.iacr.org/2011/003. Citations in this document: §4.5, §4.5.

[51] Daniel J. Bernstein, Kaushik Nath, *lib25519* (2024). URL: https://lib25519.cr.yp.to. Citations in this document: §8.2, §E.1.

[52] Daniel J. Bernstein, Peter Schwabe, *NEON crypto*, in CHES 2012 [**200**] (2012), 320–339. URL: https://cr.yp.to/papers.html#neoncrypto. Citations in this document: §E.2.

[53] Daniel J. Bernstein, Bo-Yin Yang, *Fast constant-time gcd computation and modular inversion*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019** (2019), 340–398. URL: https://tches.iacr.org/index.php/TCHES/article/view/8298. Citations in this document: §8.1.

[54] Ingrid Biehl, Bernd Meyer, Volker Müller, *Differential fault attacks on elliptic curve cryptosystems*, in Crypto 2000 [**19**] (2000), 131–146. URL: https://www.iacr.org/archive/crypto2000/18800131/18800131.pdf. Citations in this document: §9, §9.2, §A, §A.

[55] Eli Biham, Lior Neumann, *Breaking the Bluetooth pairing—the fixed coordinate invalid curve attack*, in SAC 2019 [**192**] (2019), 250–273. DOI: 10.1007/978-3-030-38471-5_11. Citations in this document: §9.2, §A.

[56] Mojtaba Bisheh-Niasar, Rami El Khatib, Reza Azarderakhsh, Mehran Mozaffari Kermani, *Fast, small, and area-time efficient architectures for key-exchange on Curve25519*, in ARITH 2020 [**2**] (2020), 72–79. DOI: 10.1109/ARITH48897.2020.00019. Citations in this document: §E.2.

[57] BITMAIN, *Bitcoin Miner S21* (2023). URL: https://shop.bitmain.com/product/detail?pid=00020240311180613891frupBW6406B2. Citations in this document: §4.1.

[58] Blockchain.com, *Total Hash Rate (TH/s)* (2024), accessed 15 June 2024. URL: https://www.blockchain.com/explorer/charts/hash-rate. Citations in this document: §7.4.

[59] Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, Eric Wustrow, *Elliptic curve cryptography in practice*, in FC 2014 [**81**] (2014), 157–175. URL: https://eprint.iacr.org/2013/734. Citations in this document: §A.

[60] Joppe W. Bos, Thorsten Kleinjung, Arjen K. Lenstra, *On the use of the negation map in the Pollard rho method*, in ANTS 2010 [**125**] (2010), 66–82. URL: https://www.joppebos.com/files/negation.pdf. Citations in this document: §4.5.

[61] Joppe W. Bos, Arjen K. Lenstra (editors), *Topics in computational number theory inspired by Peter L. Montgomery*, Cambridge University Press, 2017. ISBN 978-1107109353. See [45].

[62] Joppe W. Bos, Martijn Stam (editors), *Computational cryptography: algorithmic aspects of cryptology*, Cambridge University Press, 2021. DOI: https://doi.org/10.1017/9781108854207. See [121].

[63] Wieb Bosma, Hendrik W. Lenstra, Jr., *Complete systems of two addition laws for elliptic curves*, Journal of Number Theory **53** (1995), 229–240. MR 96f:11079. URL: https://www.math.ru.nl/~bosma/pubs/JNT1995.pdf. Citations in this document: §10.1.

[64] Colin Boyd, Paul Montague, Khanh Quoc Nguyen, *Elliptic curve based password authenticated key exchange protocols*, in [**222**] (2001), 487–501. Citations in this document: §9.6, §11.2, §11.2, §11.2.

[65] Colin Boyd, Leonie Simpson (editors), *Information security and privacy—18th Australasian conference, ACISP 2013, Brisbane, Australia, July 1–3, 2013, proceedings*, Lecture Notes in Computer Science, 7959, Springer, 2013. ISBN 978-3-642-39058-6. DOI: 10.1007/978-3-642-39059-3. See [103].

[66] ECC Brainpool, *ECC Brainpool standard curves and curve generation* (2005). URL: https://web.archive.org/web/20070814070853/http://www.ecc-brainpool.org/download/Domain-parameters.pdf. Citations in this document: §1.1, §1.3, §2.2, §2.4, §5, §7.2.

[67] Luis Brandao, Rene Peralta, *Deniable and not self-harming trapdoors*, Talk at Rump session of Crypto 2014 (2014). URL: https://crypto.2014.rump.cr.yp.to/7bad0b876b1c65ff7d0727be9afcbcf8.pdf. Citations in this document: §7.4.

[68] Gilles Brassard (editor), *Advances in cryptology—CRYPTO '89*, Lecture Notes in Computer Science, 435, Springer, Berlin, 1990. ISBN 0–387–97317–6. MR 91b:94002. See [20].

[69] Luca Breveglieri, Shay Gueron, Israel Koren, David Naccache, Jean-Pierre Seifert (editors), *Fifth international workshop on fault diagnosis and tolerance in cryptography, 2008, FDTC 2008, Washington, DC, USA, 10 August 2008*, IEEE Computer Society, 2008. ISBN 978-0-7695-3314-8. URL: https://ieeexplore.ieee.org/xpl/conhome/4599542/proceeding. See [102].

[70] Eric Brier, Marc Joye, *Weierstraß elliptic curves and side-channel attacks*, in PKC 2002 [**179**] (2002), 335–345. URL: https://marcjoye.github.io/papers/BJ02espa.pdf. Citations in this document: §8.7.

[71] Eric Brier, Marc Joye, *Fast point multiplication on elliptic curves through isogenies*, in AAECC 2003 [**101**] (2003), 43–50. URL: https://marcjoye.github.io/papers/BJ03isog.pdf. DOI: 10.1007/3-540-44828-4_6. Citations in this document: §3.1.

[72] Nicolai Brown, *Things that use Curve25519* (2024). URL: https://ianix.com/pub/curve25519-deployment.html. Citations in this document: §9.3.

[73] Nicolai Brown, *Things that use Ed25519* (2024). URL: https://ianix.com/pub/ed25519-deployment.html. Citations in this document: §9.3.

[74] Billy Bob Brumley, Risto M. Hakala, *Cache-timing template attacks*, in Asiacrypt 2009 [**170**] (2009), 667–684. URL: https://www.iacr.org/archive/asiacrypt2009/59120664/59120664.pdf. DOI: 10.1007/978-3-642-10366-7_39. Citations in this document: §8.3, §A.

[75] Billy Bob Brumley, Nicola Tuveri, *Remote timing attacks are still practical*, in ESORICS 2011 [**15**] (2011), 355–371. URL: https://eprint.iacr.org/2011/232. Citations in this document: §8.5, §A.

[76] Srdjan Capkun, Franziska Roesner (editors), *29th USENIX security symposium, USENIX Security 2020, August 12–14, 2020*, USENIX Association, 2020. ISBN 978-1-939133-17-5. URL: https://www.usenix.org/conference/usenixsecurity20. See [176].

[77] Dario Catalano, Nelly Fazio, Rosario Gennaro, Antonio Nicolosi (editors), *Public key cryptography—PKC 2011—14th international conference on practice and theory in public key cryptography, Taormina, Italy, March 6–9, 2011, proceedings*, Lecture Notes in Computer Science, 6571, Springer, 2011. ISBN 978-3-642-19378-1. See [50].

[78] Certicom, *Letter to IEEE* (2002). URL: https://www.ieee802.org/15/pub/Patent_Letters/15.3/certicom%2015.3.pdf. Citations in this document: §9.3.

[79] Liqun Chen, Shin'ichiro Matsuo (editors), *Security standardisation research—second international conference, SSR 2015, Tokyo, Japan, December*

15–16, 2015, proceedings, Lecture Notes in Computer Science, 9497, Springer, 2015. ISBN 978-3-319-27151-4. See [29].

[80] Tung Chou, *Sandy2x: New Curve25519 speed records*, in SAC 2015 [**93**] (2015), 145–160. URL: https://eprint.iacr.org/2015/943. DOI: 10.1007/978-3-319-31301-6_8. Citations in this document: §E.2.

[81] Nicolas Christin, Reihaneh Safavi-Naini (editors), *Financial cryptography and data decurity: 18th international conference, FC 2014, Christ Church, Barbados, March 3–7, 2014*, Lecture Notes in Computer Science, 8437, Springer, 2014. See [59].

[82] David V. Chudnovsky, Gregory V. Chudnovsky, *Sequences of numbers generated by addition in formal groups and new primality and factorization tests*, Advances in Applied Mathematics **7** (1986), 385–434. MR 88h:11094. Citations in this document: §3.1, §3.1, §3.1, §8, §8.1, §8.2.

[83] Henri Cohen, Atsuko Miyaji, Takatoshi Ono, *Efficient elliptic curve exponentiation using mixed coordinates*, in Asiacrypt 1998 [**186**] (1998), 51–65. URL: https://dspace02.jaist.ac.jp/dspace/bitstream/10119/4458/1/73-53.pdf. Citations in this document: §8.1.

[84] Neil Costigan, Peter Schwabe, *Fast elliptic-curve cryptography on the Cell Broadband Engine*, in Africacrypt 2009 [**198**] (2009), 368–385. URL: https://cryptojedi.org/users/peter/#celldh. Citations in this document: §E.2.

[85] Brooke Crothers, *Intel has bug-repair program* (1997). URL: https://www.cnet.com/tech/tech-industry/intel-has-bug-repair-program/. Citations in this document: §D.2.

[86] Agence nationale de la sécurité des systèmes d'information, *Publication d'un paramétrage de courbe elliptique visant des applications de passeport électronique et de l'administration électronique française* (Nov 2011). URL: https://www.ssi.gouv.fr/fr/anssi/publications/publications-scientifiques/autres-publications/publication-d-un-parametrage-de-courbe-elliptique-visant-des-applications-de.html. Citations in this document: §1.1.

[87] Yvo Desmedt (editor), *Public Key Cryptography—PKC 2003, 6th international workshop on theory and practice in public key cryptography, Miami, FL, USA, January 6–8, 2003, proceedings*, Lecture Notes in Computer Science, 2567, Springer, 2002. ISBN 3-540-00324-X. See [142].

[88] Nafissatou Diarra, Djiby Sow, Ahmed Youssef Ould Cheikh Khlil, *On indifferentiable deterministic hashing into elliptic curves*, European Journal of Pure and Applied Mathematics **10** (2017), 363–391. URL: https://ejpam.com/index.php/ejpam/article/view/2623. Citations in this document: §11.4.

[89] Claus Diem, *On the discrete logarithm problem in elliptic curves*, Compositio Mathematica **147** (2011), 75-–104. DOI: 10.1112/S0010437X10005075. Citations in this document: §2.1, §A.

[90] Claus Diem, Emmanuel Thomé, *Index calculus in class groups of non-hyperelliptic curves of genus three*, J. Cryptol. **21** (2008), 593–611. URL: https://members.loria.fr/EThome/files/non-he-genus3.pdf. Citations in this document: §7.6.

[91] Whitfield Diffie, Martin Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 644–654. ISSN 0018–9448. MR 55:10141. URL: https://ee.stanford.edu/~hellman/publications/24.pdf. Citations in this document: §1.

[92] Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, Peter Schwabe, *High-speed Curve25519 on 8-bit, 16-bit, and 32-bit microcontrollers*, Designs, Codes and Cryptography **77** (2015), 493–514. URL: https://link.springer.com/article/10.1007/s10623-015-0087-1/fulltext.html. Citations in this document: §E.2.

[93] Orr Dunkelman, Liam Keliher (editors), *Selected areas in cryptography—SAC 2015—22nd international conference, Sackville, NB, Canada, August 12–14, 2015, revised selected papers*, Lecture Notes in Computer Science, 9566, Springer, 2016. ISBN 978-3-319-31300-9. DOI: 10.1007/978-3-319-31301-6. See [80].

[94] Iwan M. Duursma, Pierrick Gaudry, François Morain, *Speeding up the discrete log computation on curves with automorphisms*, in Asiacrypt 1999 [**158**] (1999), 103–121. URL: https://inria.hal.science/inria-00511639/PDF/automorphisms.pdf. Citations in this document: §4.5.

[95] Harold M. Edwards, *A normal form for elliptic curves*, Bulletin of the American Mathematical Society **44** (2007), 393–422. URL: https://www.ams.org/bull/2007-44-03/S0273-0979-07-01153-6/home.html. Citations in this document: §3, §10.2.

[96] Adrian Escott, *Implementing a parallel Pollard rho attack on ECC* (1998). URL: https://cacr.uwaterloo.ca/conferences/1998/ecc98/escott.ps. Citations in this document: §4.5.

[97] Wolfgang Ettlinger, *Multiple vulnerabilities in Openpgp.js* (2019). URL: https://sec-consult.com/vulnerability-lab/advisory/multiple-vulnerabilities-in-openpgp-js/. Citations in this document: §9, §A.

[98] Federation of American Scientists, *Intelligence agency budgets: Commission recommends no release but releases them anyway* (1996). URL: https://irp.fas.org/commission/budget.htm. Citations in this document: §D.3.

[99] Marc Fischlin, Jean-Sébastien Coron (editors), *Advances in cryptology—EUROCRYPT 2016—35th annual international conference on the theory and applications of cryptographic techniques, Vienna, Austria, May 8–12, 2016, proceedings, part I*, Lecture Notes in Computer Science, 9665, Springer, 2016. ISBN 978-3-662-49889-7. DOI: 10.1007/978-3-662-49890-3. See [202].

[100] Agner Fog, *Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs* (2024). URL: https://agner.org/optimize/. Citations in this document: §D.1.

[101] Marc P. C. Fossorier, Tom Høholdt, Alain Poli (editors), *Applied algebra, algebraic algorithms and error-correcting codes, 15th international symposium, aaecc-15, toulouse, France, May 12–16, 2003, proceedings*, 2643, Springer, 2003. ISBN 3-540-40111-3. DOI: 10.1007/3-540-44828-4. See [71].

[102] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, Frédéric Valette, *Fault attack on elliptic curve Montgomery ladder implementation*, in FDTC 2008 [**69**] (2008), 92–98. URL: https://www.di.ens.fr/~fouque/pub/fdtc08.pdf. DOI: 10.1109/FDTC.2008.15. Citations in this document: §9.4.

[103] Pierre-Alain Fouque, Antoine Joux, Mehdi Tibouchi, *Injective encodings to elliptic curves*, in ACISP 2013 [**65**] (2013), 203–218. DOI: 10.1007/978-3-642-39059-3_14. Citations in this document: §11.4.

[104] Gerhard Frey, *How to disguise an elliptic curve (Weil descent)* (1998). URL: https://cacr.uwaterloo.ca/conferences/1998/ecc98/slides.html. Citations in this document: §2.1, §A.

[105] Gerhard Frey, Hans-Georg Rück, *A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves*, Math. Comp. **62** (1994), 865–874. URL: https://www.ams.org/journals/mcom/1994-62-206/S0025-5718-1994-1218343-6/. Citations in this document: §5.1, §A.

[106] Hayato Fujii, Diego F. Aranha, *Curve25519 for the Cortex-M4 and beyond*, in LatinCrypt 2017 [**159**] (2017), 109–127. DOI: 10.1007/978-3-030-25283-0_6. Citations in this document: §E.2.

[107] Tobias Funke, David Rupprecht, *Invalid curve attack on the 5G SUCI privacy feature* (2023). URL: https://www.gsma.com/solutions-and-impact/technologies/security/wp-content/uploads/2023/10/0073-invalid_curve.pdf. Citations in this document: §9, §9, §9, §A.

[108] Tim Güneysu, Helena Handschuh (editors), *Cryptographic hardware and embedded systems—CHES 2015—17th international workshop, Saint-Malo, France, September 13–16, 2015, proceedings*, Lecture Notes in Computer Science, 9293, Springer, 2015. ISBN 978-3-662-48323-7. See [133].

[109] Steven Galbraith, *Climbing and descending tall volcanos* (2024), ANTS 2024. URL: https://eprint.iacr.org/2024/924. Citations in this document: §7.5.

[110] Steven D. Galbraith, Mridul Nandi (editors), *Progress in cryptology—INDOCRYPT 2012, 13th international conference on cryptology in India, Kolkata, India, December 9–12, 2012, proceedings*, Lecture Notes in Computer Science, 7668, Springer, 2012. ISBN 978-3-642-34930-0. See [40].

[111] Robert P. Gallant, Robert J. Lambert, Scott A. Vanstone, *Improving the parallelized Pollard lambda search on anomalous binary curves*, Math. Comput. **69** (2000), 1699–1705. URL: https://www.ams.org/journals/mcom/2000-69-232/S0025-5718-99-01119-9/. Citations in this document: §6.2, §A.

[112] Robert P. Gallant, Robert J. Lambert, Scott A. Vanstone, *Faster point multiplication on elliptic curves with efficient endomorphisms*, in Crypto 2001 [**150**] (2001), 190–200. URL: https://iacr.org/archive/crypto2001/21390189.pdf. Citations in this document: §6.

[113] Qingguan Gao, Kaisheng Sun, Jiankuo Dong, Fangyu Zheng, Jingqiang Lin, Yongjun Ren, Zhe Liu, *V-Curve25519: efficient implementation of Curve25519 on RISC-V architecture*, in Inscrypt 2023 [**117**] (2023), 130–149. DOI: 10.1007/978-981-97-0945-8_8. Citations in this document: §E.2.

[114] Pierrick Gaudry, *Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem*, J. Symb. Comput. **44** (2009), 1690–1702. URL: https://inria.hal.science/inria-00337631. Citations in this document: §2.1, §A.

[115] Pierrick Gaudry, Emmanuel Thomé, *The mp$\mathbb{F}_q$ library and implementing curve-based key exchanges* (2007). URL: https://inria.hal.science/inria-00168429. Citations in this document: §E.2.

[116] Pierrick Gaudry, Florian Hess, Nigel P. Smart, *Constructive and destructive facets of Weil descent on elliptic curves*, Journal of Cryptology **15(1)** (2002), 19–46. URL: https://inria.hal.science/inria-00512763. Citations in this document: §2.1, §A.

[117] Chunpeng Ge, Moti Yung (editors), *Information security and cryptology—19th international conference, Inscrypt 2023, Hangzhou, China, December 9–10, 2023, revised selected papers, part II*, Lecture Notes in Computer Science, 14527, Springer, 2024. ISBN 978-981-97-0944-1. DOI: 10.1007/978-981-97-0945-8. See [113].

[118] Rosario Gennaro, Matthew Robshaw (editors), *Advances in cryptology—CRYPTO 2015—35th annual cryptology conference, Santa Barbara, CA, USA, August 16–20, 2015, proceedings, part I*, Lecture Notes in Computer Science, 9215, Springer, 2015. ISBN 978-3-662-47988-9. DOI: 10.1007/978-3-662-47989-6. See [123].

[119] Diana Goehringer, Marco Domenico Santambrogio, João M. P. Cardoso, Koen Bertels (editors), *Reconfigurable computing: architectures, tools, and applications—10th international symposium, ARC 2014, Vilamoura, Portugal, April 14–16, 2014, proceedings* (2014). ISBN 978-3-319-05959-4. See [208].

[120] Go Project, *CVE-2023-24532: Incorrect calculation on P256 curves in crypto/internal/nistec* (2023). URL: https://www.cve.org/CVERecord?id=CVE-2023-24532. Citations in this document: §10.3, §A.

[121] Robert Granger, Antoine Joux, *Computing discrete logarithms*, in [**62**] (2021), 106–139. URL: https://eprint.iacr.org/2021/1140. Citations in this document: §1.

[122] Tom R. Halfhill, *Intel ups the ante*, BYTE Magazine **FEB** (1996), 156–156. URL: https://web.archive.org/web/19981206022744/http://byte.com/art/9602/sec14/art2.htm. Citations in this document: §D.2.

[123] Mike Hamburg, *Decaf: eliminating cofactors through point compression*, in Crypto 2015 [**118**] (2015), 705–723. URL: https://eprint.iacr.org/2015/673. DOI: 10.1007/978-3-662-47989-6_34. Citations in this document: §11.5.

[124] Mike Hamburg, *Faster Montgomery and double-add ladders for short Weierstrass curves*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020** (2020), 189–208. URL: https://tches.iacr.org/index.php/TCHES/article/view/8681. Citations in this document: §8.7, §8.7.

[125] Guillaume Hanrot, François Morain, Emmanuel Thomé (editors), *Algorithmic number theory, 9th international symposium, ANTS-IX, Nancy, France, July 19–23, 2010, proceedings*, Lecture Notes in Computer Science, 6197, Springer, 2010. ISBN 978-3-642-14517-9. See [60].

[126] Feng Hao, Sushmita Ruj, Sourav Sen Gupta (editors), *Progress in cryptology—INDOCRYPT 2019—20th international conference on cryptology in India, Hyderabad, India, December 15–18, 2019, proceedings*, Lecture Notes in Computer Science, 11898, Springer, 2019. ISBN 978-3-030-35422-0. DOI: 10.1007/978-3-030-35423-7. See [211].

[127] Greg Harper, Alfred Menezes, Scott A. Vanstone, *Public-key cryptosystems with very small key length*, in Eurocrypt 1992 [**205**] (1992), 163–173. DOI: 10.1007/3-540-47555-9_14. Citations in this document: §9.3.

[128] M. Anwar Hasan, Tor Helleseth (editors), *Arithmetic of finite fields, third international workshop, WAIFI 2010, Istanbul, Turkey, June 27–30, 2010, proceedings*, Springer, 2010. ISBN 978-3-642-13796-9. See [39].

[129] Florian Hess, Sebastian Pauli, Michael E. Pohst (editors), *Algorithmic number theory, 7th international symposium, ANTS-VII, Berlin, Germany, July 23–28, 2006, proceedings*, Lecture Notes in Computer Science, 4076, Springer, 2006. ISBN 3-540-36075-1. DOI: 10.1007/11792086. See [216].

[130] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, Ed Dawson, *Twisted Edwards curves revisited*, in Asiacrypt 2008 [**196**] (2008). URL: https://eprint.iacr.org/2008/522. Citations in this document: §1.3, §3, §10.2, §10.2, §10.2, §11.5.

[131] Yvonne Hitchcock, Paul Montague, Gary Carter, Ed Dawson, *The efficiency of solving multiple discrete logarithm problems and the implications for the*

security of fixed elliptic curves, Int. J. Inf. Sec. **3** (2004), 86–98. Citations in this document: §4.2.

[132] Everett W. Howe, Kiran S. Kedlaya (editors), *ANTS X: proceedings of the tenth algorithmic number theory symposium, San Diego 2012*, Mathematical Sciences Publishers, 2013. ISBN 978-1-935107-01-9. See [41].

[133] Michael Hutter, Jürgen Schilling, Peter Schwabe, Wolfgang Wieser, *NaCl's* `crypto_box` *in hardware*, in CHES 2015 [**108**] (2015), 81–101. URL: https://cryptojedi.org/papers/#naclhw. Citations in this document: §E.2.

[134] Institute of Electrical and Electronics Engineers, *IEEE Std 1363-2000: IEEE standard specifications for public key cryptography* (2000). URL: https://perso.telecom-paristech.fr/guilley/recherche/cryptoprocesseurs/ieee/00891000.pdf. Citations in this document: §1.1, §5, §8.1.

[135] Intel, *Pentium Processor Family Developer's Manual, volume 3: Architecture and Programming Manual* (1995). URL: https://stuff.mit.edu/afs/sipb/contrib/doc/specs/ic/cpu/x86/pentium/vol3.pdf. Citations in this document: §D.1, §D.1, §D.1.

[136] Intel, *Intel delivers the next level of computing with the new Pentium II processor* (1997). URL: https://www.intel.com/pressroom/archive/releases/1997/DP050797.HTM. Citations in this document: §D.1, §D.2, §D.2.

[137] Intel, *Form 10-K* (1998). URL: https://www.intc.com/filings-reports/annual-reports/content/0000050863-98-000031/0000050863-98-000031.pdf. Citations in this document: §D.1, §D.2.

[138] Intel, *Pentium II processor at 233 MHz, 266 MHz, 300 MHz, and 333 MHz* (1998). URL: https://web.archive.org/web/20240413014621/https://datasheets.chipdb.org/Intel/x86/Pentium%20II/24333503.PDF. Citations in this document: §D.1.

[139] Intel, *Intel Architecture Optimization Reference Manual* (1999). URL: https://download.intel.com/design/PentiumII/manuals/24512701.pdf. Citations in this document: §D.1.

[140] Yuval Ishai, Vincent Rijmen (editors), *Advances in cryptology—EUROCRYPT 2019—38th annual international conference on the theory and applications of cryptographic techniques, Darmstadt, Germany, May 19–23, 2019, proceedings, part II*, Lecture Notes in Computer Science, 11477, Springer, 2019. ISBN 978-3-030-17655-6. DOI: 10.1007/978-3-030-17656-3. See [48].

[141] Tetsuya Izu, Jun Kogure, Masayuki Noro, Kazuhiro Yokoyama, *Efficient implementation of Schoof's algorithm*, in Asiacrypt 1998 [**186**] (1998), 66–79. DOI: 10.1007/3-540-49649-1_7. Citations in this document: §7.1.

[142] Tetsuya Izu, Tsuyoshi Takagi, *Exceptional procedure attack on elliptic curve cryptosystems*, in PKC 2003 [**87**] (2002), 224–239. URL: https://www.iacr.org/archive/pkc2003/25670224/25670224.pdf. Citations in this document: §10, §A.

[143] Tibor Jager, Jörg Schwenk, Juraj Somorovsky, *Practical invalid curve attacks on TLS-ECDH*, in ESORICS 2015 [**193**] (2015), 407–425. URL: https://www.nds.rub.de/research/publications/ESORICS15/. DOI: 10.1007/978-3-319-24174-6_21. Citations in this document: §A.

[144] Jan Jancar, Vladimir Sedlacek, Petr Svenda, Marek Sýs, *Minerva: the curse of ECDSA nonces: systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces*, IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020** (2020), 281–308. DOI: 10.13154/tches.v2020.i4.281-308. Citations in this document: §10, §10, §10, §A.

[145] Thomas R. Johnson, *American cryptology during the cold war, 1945–1989, Book III: retrenchment and reform, 1972–1980*, 2013, distributed inside NSA in 1998, partially declassified in 2013. URL: https://archive.org/details/cold_war_iii-nsa. Citations in this document: §D.3.

[146] Thomas R. Johnson, *American cryptology during the cold war, 1945–1989, Book IV: cryptologic rebirth, 1981–1989*, 2013, distributed inside NSA in 1999, partially declassified in 2013. URL: https://archive.org/details/cold_war_iv-nsa. Citations in this document: §D.3.

[147] Burton S. Kaliski Jr, *Elliptic curves and cryptography: A pseudorandom bit generator and other tools*, Ph.D. thesis, MIT, 1988. URL: https://dspace.mit.edu/handle/1721.1/14709. Citations in this document: §9.6.

[148] Burton S. Kaliski Jr., *A pseudo-random bit generator based on elliptic logarithms*, in Crypto 1986 [**185**] (1986), 84–103. DOI: 10.1007/3-540-47721-7_7. Citations in this document: §9.6.

[149] Burton S. Kaliski Jr. (editor), *Advances in cryptology—CRYPTO '97, 17th annual international cryptology conference, Santa Barbara, California, USA, August 17–21, 1997, proceedings*, Lecture Notes in Computer Science, 1294, Springer, 1997. ISBN 3-540-63384-7. See [165].

[150] Joe Kilian (editor), *Advances in cryptology—CRYPTO 2001, 21st annual international cryptology conference, Santa Barbara, California, USA, August 19–23, 2001, proceedings*, Lecture Notes in Computer Science, 2139, Springer, 2001. ISBN 3-540-42456-3. See [112].

[151] Ann Hibner Koblitz, Neal Koblitz, Alfred Menezes, *Elliptic curve cryptography: The serpentine course of a paradigm shift*, Journal of Number Theory **131** (2011), 781–814. URL: https://eprint.iacr.org/2008/390. Citations in this document: §7.6.

[152] Neal Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), 203–209. ISSN 0025–5718. MR 88b:94017. URL: https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866109-5/. Citations in this document: §1.1.

[153] Neal Koblitz, Alfred Menezes, *A riddle wrapped in an enigma*, IEEE Security & Privacy **14** (2016), 34–42. URL: https://eprint.iacr.org/2015/1018. DOI: 10.1109/MSP.2016.120. Citations in this document: §7.1, §7.4, §7.4, §7.5, §7.5, §D.

[154] Maurice Kraitchik, *Théorie des nombres I*, Gauthier-Villars, 1922. URL: https://archive.org/details/thoriedesnombres01krai. Citations in this document: §1.

[155] Maurice Kraitchik, *Théorie des nombres II*, Gauthier-Villars, 1926. URL: https://archive.org/details/thoriedesnombres02krai. Citations in this document: §1.

[156] Fabian Kuhn, Rene Struik, *Random walks revisited: extensions of Pollard's rho algorithm for computing multiple discrete logarithms*, in [**224**] (2001), 212–229. URL: https://www.distcomp.ethz.ch/publications.html. Citations in this document: §4.2.

[157] Kaoru Kurosawa (editor), *Advances in cryptology—ASIACRYPT 2007, 13th international conference on the theory and application of cryptology and information security, Kuching, Malaysia, December 2–6, 2007, proceedings*, Lecture Notes in Computer Science, 4833, Springer, 2007. ISBN 978-3-540-76899-9. See [36].

[158] Kwok-Yan Lam, Eiji Okamoto, Chaoping Xing (editors), *Advances in cryptology—ASIACRYPT '99, international conference on the theory and applications of cryptology and information security, Singapore, November 14–18, 1999, proceedings*, Lecture Notes in Computer Science, 1716, Springer, 1999. ISBN 3-540-66666-4. See [94].

[159] Tanja Lange, Orr Dunkelman (editors), *Progress in cryptology—LATINCRYPT 2017—5th international conference on cryptology and information security in Latin America, Havana, Cuba, September 20–22, 2017, revised selected papers*, Lecture Notes in Computer Science, 11368, Springer, 2019. ISBN 978-3-030-25282-3. DOI: 10.1007/978-3-030-25283-0. See [106].

[160] Adam Langley, Andrew Moon, *Implementations of a fast elliptic-curve Digital Signature Algorithm* (2013). URL: https://github.com/floodyberry/ed25519-donna. Citations in this document: §E.2.

[161] Kristin E. Lauter, Francisco Rodríguez-Henríquez (editors), *Progress in cryptology—LATINCRYPT 2015—4th international conference on cryptology and information security in Latin America, Guadalajara, Mexico, August 23–26, 2015, proceedings*, Lecture Notes in Computer Science, 9230, Springer, 2015. ISBN 978-3-319-22173-1. DOI: 10.1007/978-3-319-22174-8. See [30].

[162] Hyung Tae Lee, Jung Hee Cheon, Jin Hong, *Accelerating ID-based encryption based on trapdoor DL using pre-computation* (2011). URL: https://eprint.iacr.org/2011/187. Citations in this document: §4.2.

[163] Donald J. Lewis (editor), *1969 Number Theory Institute*, Proceedings of Symposia in Pure Mathematics, 20, American Mathematical Society, Providence, Rhode Island, 1971. ISBN 0–8218–1420–6. MR 47:3286. See [217].

[164] Jay Ligatti, Xinming Ou, Jonathan Katz, Giovanni Vigna (editors), *CCS '20: 2020 ACM SIGSAC conference on computer and communications security, virtual event, USA, November 9–13, 2020*, ACM, 2020. ISBN 978-1-4503-7089-9. DOI: 10.1145/3372297. See [13].

[165] Chae Hoon Lim, Pil Joong Lee, *A key recovery attack on discrete log-based schemes using a prime order subgroup*, in Crypto 1997 [**149**] (1997), 249–263. DOI: 10.1007/BFb0052240. Citations in this document: §9.1.

[166] Luigi1111, Riccardo Spagni, *Disclosure of a major bug in CryptoNote based currencies* (2017). URL: https://www.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html. Citations in this document: §11.5.

[167] Eric M. Mahé, Jean-Marie Chauvet, *Fast GPGPU-based elliptic curve scalar multiplication* (2014). URL: https://eprint.iacr.org/2014/198. Citations in this document: §E.2.

[168] Stefan Mangard, François-Xavier Standaert (editors), *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th international workshop, Santa Barbara, CA, USA, August 17–20, 2010, proceedings*, Lecture Notes in Computer Science, 6225, Springer, 2010. ISBN 978-3-642-15030-2. See [10].

[169] Dimitri Mankowski, Thom Wiggers, Veelasha Moonsamy, *TLS → post-quantum TLS: inspecting the TLS landscape for PQC adoption on Android*, in EuroSPW [**5**] (2023), 526–538. DOI: 10.1109/EuroSPW59978.2023.00065. Citations in this document: §9.3.

[170] Mitsuru Matsui (editor), *Advances in cryptology—ASIACRYPT 2009, 15th international conference on the theory and application of cryptology and information security, Tokyo, Japan, December 6–10, 2009, proceedings*, Lecture Notes in Computer Science, 5912, Springer, 2009. ISBN 978-3-642-10365-0. DOI: 10.1007/978-3-642-10366-7. See [74].

[171] Mohamad Ali Mehrabi, Christophe Doche, *Low-cost, low-power FPGA implementation of ED25519 and CURVE25519 point multiplication*, Inf. **10** (2019), 285. DOI: 10.3390/info10090285. Citations in this document: §E.2.

[172] Alfred J. Menezes, Tatsuaki Okamoto, Scott A. Vanstone, *Reducing elliptic curve logarithms to a finite field*, IEEE Transactions on Information Theory **39** (1993), 1639–1646. DOI: 10.1145/103418.103434. Citations in this document: §5.1, §A.

[173] Microsoft, *Windows CryptoAPI spoofing vulnerability* (2020). URL: https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2020-0601. Citations in this document: §7.8, §A.

[174] Victor S. Miller, *Use of elliptic curves in cryptography*, in Crypto 1985 [**229**] (1986), 417–426. MR 88b:68040. DOI: 10.1007/3-540-39799-X_31. Citations in this document: §1.1, §8.7, §8.7, §11.4.

[175] MITRE, *CVE-2023-46324* (2023). URL: https://www.cve.org/CVERecord?id=CVE-2023-46324. Citations in this document: §9.

[176] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, Nadia Heninger, *TPM-FAIL: TPM meets timing and lattice attacks*, in USENIX Security 2020 [**76**] (2020), 2057–2073. URL: https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm. Citations in this document: §10, §A.

[177] Peter L. Montgomery, *Speeding the Pollard and elliptic curve methods of factorization*, Mathematics of Computation **48** (1987), 243–264. ISSN 0025–5718. MR 88e:11130. URL: https://www.ams.org/journals/mcom/1987-48-177/S0025-5718-1987-0866113-7/. Citations in this document: §1.3, §3, §8, §8, §8.2.

[178] Mozilla Foundation, *Security Advisory 2023-56* (2023). URL: https://www.mozilla.org/en-US/security/advisories/mfsa2023-56/. Citations in this document: §10, §A.

[179] David Naccache, Pascal Paillier (editors), *Public Key Cryptography, 5th international workshop on practice and theory in public key cryptosystems, PKC 2002, Paris, France, February 12–14, 2002, proceedings*, Lecture Notes in Computer Science, 2274, Springer, 2002. ISBN 3-540-43168-3. See [70].

[180] Kaushik Nath, Palash Sarkar, *Efficient arithmetic in (pseudo-)Mersenne prime order fields*, Adv. Math. Commun. **16** (2022), 303–348. DOI: 10.3934/amc.2020113. Citations in this document: §E.2.

[181] Kaushik Nath, Palash Sarkar, *Security and efficiency trade-offs for elliptic curve Diffie-Hellman at the 128-bit and 224-bit security levels*, J. Cryptogr. Eng. **12** (2022), 107–121. DOI: 10.1007/s13389-021-00261-y. Citations in this document: §E.2.

[182] Kaushik Nath, Palash Sarkar, *Efficient 4-way vectorizations of the Montgomery ladder*, IEEE Trans. Computers **71** (2022), 712–723. DOI: 10.1109/TC.2021.3060505. Citations in this document: §E.2.

[183] National Institute of Standards and Technology, *Request for Comments on FIPS 186-5 and SP 800-186*, Federal Register **84** (2019), 58373–58375. URL: https://www.federalregister.gov/documents/2019/10/31/2019-23742/request-for-comments-on-fips-186-5-and-sp-800-186. Citations in this document: §1.4, §1.4, §1.4.

[184] National Security Agency, *Fact Sheet NSA Suite B Cryptography* (2005). URL: https://web.archive.org/web/20051125141648/https://www.nsa.gov/ia/industry/crypto_suite_b.cfm. Citations in this document: §1.1.

[185] Andrew M. Odlyzko (editor), *Advances in cryptology—CRYPTO '86, Santa Barbara, California, USA, 1986, proceedings*, Lecture Notes in Computer Science, 263, Springer, 1987. See [148].

[186] Kazuo Ohta, Dingyi Pei (editors), *Advances in cryptology—ASIACRYPT '98, International conference on the theory and applications of cryptology and information security, Beijing, China, October 18–22, 1998, proceedings*, Lecture Notes in Computer Science, 1514, Springer, 1998. ISBN 3-540-65109-8. DOI: 10.1007/3-540-49649-1. See [83], [141].

[187] Paul C. van Oorschot, Michael Wiener, *Parallel collision search with cryptanalytic applications*, Journal of Cryptology **12** (1999), 1–28. ISSN 0933–2790. DOI: 10.1007/PL00003816. Citations in this document: §4.4, §4.5.

[188] OpenSSL Project, *Bignum squaring may produce incorrect results (CVE-2014-3570)* (2015). URL: https://www.openssl.org/news/secadv/20150108.txt. Citations in this document: §A, §A.

[189] OpenSSL Project, *OpenSSL 3.2.2* (2024). URL: https://www.openssl.org/source/. Citations in this document: §E.1.

[190] Elisabeth Oswald, Pankaj Rohatgi (editors), *Cryptographic hardware and embedded systems—CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10–13, 2008, Proceedings*, Lecture Notes in Computer Science, 5154, Springer, 2008. ISBN 978-3-540-85052-6. See [47].

[191] Sarvar Patel, *Number theoretic attacks on secure password schemes*, in S&P 1997 [**1**] (1997), 236–247. DOI: 10.1109/SECPRI.1997.601340. Citations in this document: §11.2.

[192] Kenneth G. Paterson, Douglas Stebila (editors), *Selected areas in cryptography—SAC 2019—26th international conference, Waterloo, ON, Canada, August 12–16, 2019, revised selected papers*, Lecture Notes in Computer Science, 11959, Springer, 2020. ISBN 978-3-030-38470-8. DOI: 10.1007/978-3-030-38471-5. See [55].

[193] Günther Pernul, Peter Y. A. Ryan, Edgar R. Weippl (editors), *Computer security—ESORICS 2015—20th European symposium on research in computer security, Vienna, Austria, September 21–25, 2015, proceedings, part I*, Lecture Notes in Computer Science, 9326, Springer, 2015. ISBN 978-3-319-24173-9. DOI: 10.1007/978-3-319-24174-6. See [143].

[194] Léo Perrin, *Partitions in the S-box of Streebog and Kuznyechik*, IACR Trans. Symmetric Cryptol. **2019** (2019), 302–329. URL: https://who.paris.inria.fr/Leo.Perrin/pi.html. Citations in this document: §7.

[195] Christophe Petit, Jean-Jacques Quisquater, *On polynomial systems arising from a Weil descent*, in Asiacrypt 2012 [**225**] (2012), 451–466. URL: https://www.iacr.org/archive/asiacrypt2012/76580446/76580446.pdf. Citations in this document: §2.1, §A.

[196] Josef Pieprzyk (editor), *Advances in cryptology — ASIACRYPT 2008, 14th international conference on the theory and application of cryptology and information security, Melbourne, Australia, December 7–11, 2008*, Lecture Notes in Computer Science, 5350, 2008. ISBN 978-3-540-89254-0. See [130].

[197] John M. Pollard, *Monte Carlo methods for index computation mod p*, Mathematics of Computation **32** (1978), 918–924. URL: https://www.ams.org/journals/mcom/1978-32-143/S0025-5718-1978-0491431-9/. Citations in this document: §4, §4.5, §4.5, §9.1.

[198] Bart Preneel (editor), *Progress in cryptology—AFRICACRYPT 2009, second international conference on cryptology in Africa, Gammarth, Tunisia, June 21–25, 2009, proceedings*, Lecture Notes in Computer Science, 5580, Springer, 2009. See [84].

[199] Bart Preneel, Tsuyoshi Takagi (editors), *Cryptographic Hardware and Embedded Systems—CHES 2011—13th international workshop, Nara, Japan, September 28–October 1, 2011, proceedings*, Lecture Notes in Computer Science, 6917, Springer, 2011. ISBN 978-3-642-23950-2. See [31].

[200] Emmanuel Prouff, Patrick Schaumont (editors), *Cryptographic hardware and embedded systems—CHES 2012—14th international workshop, Leuven, Belgium, September 9–12, 2012, proceedings*, Lecture Notes in Computer Science, 7428, Springer, 2012. ISBN 978-3-642-33026-1. See [52].

[201] Red Hat, *CVE-2021-3798* (2022). URL: https://www.cve.org/CVERecord?id=CVE-2021-3798. Citations in this document: §9, §A, §A.

[202] Joost Renes, Craig Costello, Lejla Batina, *Complete addition formulas for prime order elliptic curves*, in Eurocrypt 2016 [**99**] (2016), 403–428. URL: https://eprint.iacr.org/2015/1060. DOI: 10.1007/978-3-662-49890-3_16. Citations in this document: §10.3, §10.3, §10.3, §10.3.

[203] Certicom Research, *SEC 2: recommended elliptic curve domain parameters, version 1.0* (2000). URL: https://www.secg.org/SEC2-Ver-1.0.pdf. Citations in this document: §1.1, §5, §7.1.

[204] Ronald L. Rivest, Martin E. Hellman, John C. Anderson, John W. Lyons, *Responses to NIST's proposal*, Communications of the ACM **35** (1992), 41–54. URL: https://people.csail.mit.edu/rivest/pubs/RHAL92.pdf. Citations in this document: §1.4.

[205] Rainer A. Rueppel (editor), *Advances in cryptology—EUROCRYPT '92, workshop on the theory and application of of cryptographic techniques, Balatonfüred, Hungary, May 24–28, 1992, proceedings*, Lecture Notes in Computer Science, 658, Springer, 1993. ISBN 3-540-56413-6. DOI: 10.1007/3-540-47555-9. See [127].

[206] Peter Y. A. Ryan, David Naccache, Jean-Jacques Quisquater (editors), *The New Codebreakers—Essays dedicated to David Kahn on the occasion of his 85th birthday*, Lecture Notes in Computer Science, 9100, Springer, 2016. ISBN 978-3-662-49300-7. See [49].

[207] Ahmad-Reza Sadeghi, Virgil D. Gligor, Moti Yung (editors), *2013 ACM SIGSAC conference on computer and communications security, CCS'13, Berlin, Germany, November 4–8, 2013*, ACM, 2013. ISBN 978-1-4503-2477-9. See [34].

[208] Pascal Sasdrich, Tim Güneysu, *Efficient elliptic-curve cryptography using Curve25519 on reconfigurable devices*, in ARC 2014 [**119**] (2014), 25–36. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=ddbd39a27e4db00813a4c5d2da0e19d6fb50bbcb. Citations in this document: §E.2.

[209] Takakzu Satoh, Kiyomichi Araki, *Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves*, Commentarii Mathematici Universitatis Sancti Pauli **47** (1998), 81–92. URL: https://rikkyo.repo.nii.ac.jp/records/9910. Citations in this document: §5.1, §A.

[210] Stephen Schwab, William K. Robertson, Davide Balzarotti (editors), *Proceedings of the 32nd annual conference on computer security applications, ACSAC 2016, Los Angeles, CA, USA, December 5–9, 2016*, ACM, 2016. ISBN 978-1-4503-4771-6. DOI: 10.1145/2991079. See [12].

[211] Peter Schwabe, Amber Sprenkels, *The complete cost of cofactor $h = 1$*, in INDOCRYPT 2019 [**126**] (2019), 375–397. URL: https://eprint.iacr.org/2019/1166. DOI: 10.1007/978-3-030-35423-7_19. Citations in this document: §10.3.

[212] Michael Scott, *Re: NIST annouces set of Elliptic Curves.* (1999). URL: https://groups.google.com/g/sci.crypt/c/mFMukSsORmI/m/FpbHDQ6hM_MJ. Citations in this document: §7.1.

[213] Igor A. Semaev, *On computing logarithms on elliptic curves*, Diskretnaya Matematika **8** (1996), 65–71, in Russian; see also newer version [**214**]. DOI: 10.4213/dm516.

[214] Igor A. Semaev, *On computing logarithms on elliptic curves*, Discrete Mathematics and Applications **6** (1996), 69–76; see also older version [**213**]. DOI: 10.1515/dma.1996.6.1.69. Citations in this document: §5.1, §A.

[215] Igor A. Semaev, *Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p*, Mathematics of Computation **67** (1998), 353–356. URL: https://www.ams.org/journals/mcom/1998-67-221/S0025-5718-98-00887-4/. Citations in this document: §5.1, §A.

[216] Andrew Shallue, Christiaan E. van de Woestijne, *Construction of rational points on elliptic curves over finite fields*, in ANTS 2006 [**129**] (2006), 510–524. DOI: 10.1007/11792086_36. Citations in this document: §11.4.

[217] Daniel Shanks, *Class number, a theory of factorization, and genera*, in [**163**] (1971), 415–440. MR 47:4932. Citations in this document: §4.5, §4.5.

[218] Nigel P. Smart, *The discrete logarithm problem on elliptic curves of trace one*, Journal of Cryptology **12(3)** (1999), 193–196. DOI: 10.1007/s001459900052. Citations in this document: §5.1, §A.

[219] David L. Sobel, *New NIST/NSA Revelations* (1993). URL: https://web.archive.org/web/20200229145033/https://catless.ncl.ac.uk/Risks/14/59#subj7. Citations in this document: §7.

[220] National Institute for Standards and Technology, *Digital signature standard*, Federal Information Processing Standards Publication **186-2** (2000). URL: https://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf. Citations in this document: §1.1, §1.3, §7.1.

[221] Mathy Vanhoef, Eyal Ronen, *Dragonblood: analyzing the Dragonfly handshake of WPA3 and EAP-pwd*, in S&P 2020 [**3**] (2020), 517–533. URL: https://eprint.iacr.org/2019/383. DOI: 10.1109/SP40000.2020.00031. Citations in this document: §11.3, §11.3, §11.3, §11.3, §A.

[222] Vijay Varadharajan, Yi Mu (editors), *Information security and privacy, 6th Australasian conference, ACISP 2001, Sydney, Australia, July 11–13, 2001, proceedings*, Lecture Notes in Computer Science, 2119, Springer, 2001. ISBN 3-540-42300-1. See [64].

[223] Serge Vaudenay (editor), *Progress in Cryptology—AFRICACRYPT 2008, First international conference on cryptology in Africa, Casablanca, Morocco, June 11–14, 2008, proceedings*, Lecture Notes in Computer Science, 5023, Springer, 2008. ISBN 978-3-540-68159-5. See [27].

[224] Serge Vaudenay, Amr M. Youssef (editors), *Selected Areas in Cryptography: 8th annual international workshop, SAC 2001, Toronto, Ontario, Canada, August 16–17, 2001, revised papers*, Lecture Notes in Computer Science, 2259, Springer, 2001. ISBN 3–540–43066–0. MR 2004k:94066. See [156].

[225] Xiaoyun Wang, Kazue Sako (editors), *Advances in cryptology—ASIACRYPT 2012—18th international conference on the theory and application of cryptology and information security, Beijing, China, December 2–6, 2012, proceedings*, Lecture Notes in Computer Science, 7658, Springer, 2012. ISBN 978-3-642-34960-7. See [195].

[226] A.E. Western, Jeffery Charles Percy Miller, *Tables of indices and primitive roots* (1968). URL: https://books.google.com/books?id=amw5wwEACAAJ. Citations in this document: §1.

[227] Zack Whittaker, *New leaked documents detail secret U.S. intelligence 'black budget' figures* (2013). URL: https://www.zdnet.com/article/new-leaked-documents-detail-secret-u-s-intelligence-black-budget-figures/. Citations in this document: §D.3.

[228] Michael J. Wiener, Robert J. Zuccherato, *Faster attacks on elliptic curve cryptosystems* (1998). URL: https://grouper.ieee.org/groups/1363/Research/contributions/attackEC.ps. Citations in this document: §4.5, §A.

[229] Hugh C. Williams (editor), *Advances in cryptology: CRYPTO '85*, Lecture Notes in Computer Science, 218, Springer, Berlin, 1986. ISBN 3–540–16463–4. See [174].

[230] Guiming Wu, Qianwen He, Jiali Jiang, Zhenxiang Zhang, Xin Long, Yuan Zhao, Yinchao Zou, *A high-performance hardware architecture for ECC point multiplication over Curve25519*, in FCCM 2022 [4] (2022), 1–9. DOI: 10.1109/FCCM53951.2022.9786192. Citations in this document: §E.2.

[231] Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, Tal Malkin (editors), *Public Key Cryptography—PKC 2006—9th international conference on theory and practice in public-key cryptography, New York, NY, USA, April 24–26, 2006, proceedings*, Lecture Notes in Computer Science, 3958, Springer, 2006. ISBN 978-3-540-33851-2. See [23].

A Chronology of ECC vulnerabilities

This appendix reviews the timeline of ECC vulnerabilities. In this appendix, "demo" means that an attack has been demonstrated; "vulnerability" means that analysis indicates that an attack should work; "potential vulnerability" means that the attacker has extra power but further analysis is required to determine whether there is a vulnerability; "speedup" means that attacks are faster but not necessarily feasible.

This appendix focuses on the core ECC tasks, namely ECDH and signatures; this excludes, e.g., Dual EC (see [49]) and many breaks of advanced protocols built using ECC. For side-channel attacks, this appendix focuses on timing attacks and excludes power attacks, electromagnetic attacks, etc.

The timeline is organized by publication date. Two types of publications are included: publications pointing out new attack strategies (e.g., [54] pointing out invalid-curve vulnerabilities), and publications pointing out vulnerabilities in specific implementations (e.g., [201] pointing out an invalid-curve vulnerability in openCryptoki). The timeline is as follows:

- 1993 [172], independently 1994 [105], independently 1996 [214]: multiplicative transfers against curves with $p-1$ points, $p+1$ points, etc.
- 1998 [215], independently 1998 [209], independently 1999 [218]: additive transfers against curves with p points.
- 1998 [104]: sketch of speedup against some curves over non-prime fields.
- 1998 [228]: negation speedup.

- 2000 [111]: speedup using other fast endomorphisms for some curves.
- 2000 [54]: invalid-curve vulnerabilities in general.
- 2002 [116]: speedup against some curves over non-prime fields.
- 2002 [142]: exceptional-procedure vulnerabilities in general.
- 2009 [114]: speedup against some curves over non-prime fields.
- 2009 [74]: cache-timing demo against OpenSSL (0.9.8k and under).
- 2010 [6]: repeated-nonce demo extracting the Sony PlayStation 3 ECDSA signing key.
- 2011 [89]: speedup against some curves over non-prime fields.
- 2011 [75]: timing demo extracting NIST B-163 ECDSA secret keys from OpenSSL.
- 2012 [195]: speedup against some curves over non-prime fields.
- 2013 [59]: repeated-nonce demo extracting some Bitcoin ECDSA secret keys.
- 2015 [188]: potential invalid-curve vulnerability in OpenSSL because of an arithmetic bug inside a point-on-curve test. The arithmetic bug applies to occasional inputs, and [188] says the "exact impact is difficult to determine".
- 2015 [12]: timing demo extracting secp256k1 ECDSA secret keys from OpenSSL.
- 2015 [143]: invalid-curve demo extracting TLS ECDH secret keys from 2 out of 8 analyzed libraries: "Oracle's default Java TLS implementation (JSSE with a SunEC provider) and TLS servers using the Bouncy Castle library)".
- 2019 [97]: invalid-curve vulnerability in OpenPGP.js.
- 2019 [144]: "Minerva" timing demo extracting ECDSA secret keys from a FIPS-certified CC-certified Athena IDProtect smart card. Same vulnerability in seven other certified devices, and in 4 out of 13 software libraries.
- 2019 [176]: "TPM-FAIL" timing demo extracting ECDSA secret keys from two certified TPMs.
- 2019 [55]: invalid-curve demo against Bluetooth pairing.
- 2019 [221]: "Dragonblood" demo against the WPA3 Dragonfly handshake, including invalid-curve attacks and timing attacks.
- 2020 [173]: "CurveBall" parameter-substitution vulnerability in ECDSA in Windows 10.
- 2020 [11]: timing demo extracting ECDSA secret keys from mbedTLS.
- 2020 [13]: "LadderLeak" timing demo extracting ECDSA secret keys from OpenSSL in some scenarios.
- 2022 [201]: invalid-curve vulnerability in the openCryptoki soft token.
- 2023 [120]: potential exceptional-procedure vulnerability for NIST P-256 in the crypto/elliptic library for Go.
- 2023 [107]: invalid-curve vulnerability for 5G Subscription Concealed Identifiers in the free5GC udm software.
- 2023 [178]: timing vulnerability for NIST curves in the NSS cryptographic library in the Firefox browser.

B Bit operations for elliptic-curve discrete logarithms

This appendix reviews the number of bit operations used in state-of-the-art ECDLP attacks for curves meeting the SafeCurves criteria. Note that faster attacks are known against various curves not meeting the SafeCurves criteria.

This appendix takes Curve25519 as a concrete example, with $p = 2^{255} - 19$ and $\ell \approx 2^{252}$. It was stated in [23] that every attack known at that time was "more expensive than performing a brute-force search on a typical 128-bit secret-key cipher"; this is also true for every attack known today.

Structurally, this appendix counts the number of iterations in an attack (Appendix B.1), counts the number of multiplications in each iteration (Appendix B.2), and takes optimized bit-operation counts from [48] for each multiplication (Appendix B.3). The reader is cautioned that this analysis omits two small effects in opposite directions: first, there is overhead in each iteration beyond multiplications; second, the operation counts from [48] can still be improved. See Appendix B.3 for further details. It would be interesting to analyze the exact number of bit operations for an optimized iteration, accounting for these improvements and for all overheads. See our paper [39] (and, for more background, [16]) for a detailed analysis of bit-operation counts for ECC2K-130, a binary-field ECDL challenge.

B.1 Iterations

As mentioned in Section 4, a standard negating rho attack takes about $\sqrt{\pi\ell/4}$ iterations on average: e.g., $2^{125.8}$ iterations for Curve25519.

We showed in [41] that the constant $\sqrt{\pi/4} \approx 0.886$ is not optimal in cost models that allow free memory access. Perhaps an improvement is possible even when one counts the bit operations involved in memory access. In the generic-group model augmented with free negation, the constant cannot be better than $2/3 \approx 0.667$; e.g., $2^{125.4}$ iterations for Curve25519.

For comparison, brute-force search on a 128-bit secret-key cipher takes about 2^{127} iterations on average, but the iterations are typically much less expensive in the cipher case. For example, [28] shows that a complete AES-128 attack iteration uses under $2^{14.9}$ bit operations, whereas Appendices B.2 and B.3 below indicate that the main operations in a Curve25519 attack iteration are close to 2^{20} bit operations. Presumably the costs of routing data will also be larger for a Curve25519 attack iteration than for an AES-128 attack iteration, but this appendix focuses on bit operations. See also Sections 4.2 and 4.3 regarding multi-target attacks.

B.2 Multiplications per iteration

The main work in an iteration inside the standard negating rho attack is an addition on a short Weierstrass curve in affine coordinates. Addition in affine coordinates involves a division, but the division is batched across many parallel attack iterations, reducing the effective cost of each division to $4\mathbf{M}$ and the total cost of each iteration to $6\mathbf{M}$.

The details are as follows. On the curve $y^2 = x^3 + ax + b$, the sum of two points (x_1, y_1) and (x_2, y_2) is (x_3, y_3) where $\lambda = (y_2 - y_1)/(x_2 - x_1)$, $x_3 = \lambda^2 - x_1 - x_2$, and $y_3 = (x_1 - x_3)\lambda - y_1$. Failure cases in these formulas (see Section 10) are not relevant to the attack.

The inversion of $x_2 - x_1$ costs $3\mathbf{M}$ as part of a large batch. There is then $1\mathbf{M}$ to multiply by $y_2 - y_1$, $1\mathbf{M}$ to square λ, and $1\mathbf{M}$ to multiply by $x_1 - x_3$.

B.3 Bit operations per multiplication

Asymptotically, each multiplication in $\mathbb{F}_p$ uses $(\log p)^{1+o(1)}$ bit operations. For concrete numbers, we focus on the case of Curve25519.

An unrolled circuit for 255-bit integer multiplication, including Karatsuba's method and many lower-layer speedups, uses 173954 bit operations, according to the software from [48]. A 255-bit squaring circuit uses 103000 bit operations, according to the same software. Five multiplications and a squaring with these circuits use $972770 \approx 2^{19.89}$ bit operations.

As noted above, this analysis omits some known improvements. Karatsuba's method saves time when inputs are reused across multiplications; batched divisions reuse some inputs. Also, [33, footnote 5] points out that iteration details can be set up to guarantee that some trailing bits of $x_2 - x_1$ are zero, saving time in multiplications. As a possible further improvement, more advanced multiplication methods such as Toom's method asymptotically outperform Karatsuba's method, and perhaps already save operations for 255-bit inputs.

In the opposite direction, these are just the multiplication costs. There are also some costs for subtractions, reductions mod $2^{255} - 19$, "distinguished point" management, etc.

C An email exchange

```
Subject: Greetings from evil Jerry
From: "Jerome A. Solinas" <jasolin@tycho.ncsc.mil>
Date: Wed, 23 Jul 2014 13:49:32 -0400
To: djb@cr.yp.to


Dr. Dan,

  Enjoyed your new paper.  Now I can cross "become an evil Internet meme"
off my bucket list.

  I will be at PQC Waterloo, so I'll see you and Tanja there if you're
going.


Regards,

-- Jerry


Subject: Re: Greetings from evil Jerry
From: "D. J. Bernstein" <djb@cr.yp.to>
```

Date: 6 Sep 2014 08:54:43 -0000
To: "Jerome A. Solinas" <jasolin@tycho.ncsc.mil>

Hi Jerry,

Unfortunately we won't be able to make it to PQCrypto. Maybe we'll see you at the NIST events in the Spring.

Out of curiosity, where _did_ you get the seeds for NIST P-256 etc.?

---Dan

Subject: Re: Greetings from evil Jerry
From: "Jerome A. Solinas" <jasolin@tycho.ncsc.mil>
Date: Tue, 16 Sep 2014 14:53:14 -0400
To: "D. J. Bernstein" <djb@cr.yp.to>

On 09/06/2014 04:54 AM, D. J. Bernstein wrote:
> Hi Jerry,
>
> Unfortunately we won't be able to make it to PQCrypto. Maybe we'll see
> you at the NIST events in the Spring.
>
> Out of curiosity, where _did_ you get the seeds for NIST P-256 etc.?
>
> ---Dan

Interesting question. We built all the seeds via hashing (SHA-1, I think) from the ASCII representation of a humorous message. Unfortunately, we can remember neither the (exact) message nor the details of how we hashed. Too bad, since we could prove our innocence by disclosing the details, if only we could remember them.

-- j

Subject: Re: Greetings from evil Jerry
From: "D. J. Bernstein" <djb@cr.yp.to>
Date: 14 Jun 2015 03:15:54 -0000
To: "Jerome A. Solinas" <jasolin@tycho.ncsc.mil>

Jerome A. Solinas writes, back in September:
> We built all the seeds via hashing (SHA-1, I think) from the ASCII
> representation of a humorous message. Unfortunately, we can remember
> neither the (exact) message nor the details of how we hashed.

Do you have some examples of similar messages? Kevin already mentioned that it was something along the lines of "Jerry will get a raise", but the next question is where you would have put a counter (or could it

have been repeated hashing?) to try multiple seeds. I'd be happy to
throw some cluster time at this.

---Dan

Subject: Re: Greetings from evil Jerry
From: "Jerome A. Solinas" <jasolin@tycho.ncsc.mil>
Date: Wed, 17 Jun 2015 10:27:34 -0400
To: "D. J. Bernstein" <djb@cr.yp.to>

On 06/13/2015 11:15 PM, D. J. Bernstein wrote:
> Jerome A. Solinas writes, back in September:
> > We built all the seeds via hashing (SHA-1, I think) from the ASCII
> > representation of a humorous message. Unfortunately, we can remember
> > neither the (exact) message nor the details of how we hashed.
> Do you have some examples of similar messages? Kevin already mentioned
> that it was something along the lines of "Jerry will get a raise", but
> the next question is where you would have put a counter (or could it
> have been repeated hashing?) to try multiple seeds. I'd be happy to
> throw some cluster time at this.
>
> ---Dan

I believe there was a counter rather than multiple hashing, but I don't know
details. The message was along the lines of "Give Bob and Jerry a raise" or
"Bob and Jerry rule" or something like that. It was Bob Reiter who actually
wrote the code, and he doesn't remember the details either. Nor were we
able to find archives from so long ago. If they exist, they are no doubt
sitting on a hard drive near the Ark of the Covenant.

I know this isn't much to go on. We really didn't think it would ever
matter.

-- Jerry

D Chips available to attackers

This appendix investigates the claim from [**153**, 2018 eprint version] that "2^{86} bit operations ... almost certainly was beyond the NSA's capacity in 1997".

D.1 An example of CPU efficiency

Consider the 266 MHz Intel Pentium II. This CPU was released in May 1997, according to [**136**] and [**137**, pages 2–3]. The CPU carried out "one FMUL per two clock cycles", according to [**100**, page 83] (see also [**139**, page 2-30]); i.e., the CPU carried out 133 million FMUL instructions per second.

Each FMUL instruction multiplies two inputs in "FPU registers" (see, e.g., [135, page 25-117]), each of which is a floating-point number in "extended format" (see, e.g., [135, page 6-25]), meaning an 80-bit floating-point number with a 64-bit mantissa (see, e.g., [135, page 6-23]). The software from [48] reports 17402 bit operations for 64-bit integer multiplication. The CPU was also carrying out many further bit operations for instruction decoding, out-of-order execution, etc. We estimate at least 2^{15} bit operations in total per FMUL, and thus at least 2^{42} bit operations per second for the CPU.

Intel's data sheet for the processor [138, page 22] indicated that the CPU core would draw at most 12.7 A at 2.8 V, plus at most 1.44 A at 3.3 V for L2 cache; i.e., at most 40 watts. We estimate 2^6 watts to account for supporting circuitry, power-supply inefficiency, etc., and conclude that this CPU was carrying out at least 2^{36} bit operations per joule.

For comparison, the current Bitcoin-mining equipment cited in Section 4.1 carries out 2^{54} bit operations per joule. This indicates about 1.5 years per doubling of energy efficiency of a bit operation. Interpolating would suggest 2^{47} bit operations per joule in 2013, in line with our estimate [43] at the time.

D.2 Scaling to many chips

Given the above estimate of at least 2^{36} bit operations per joule with 1997 chip technology, 2^{86} bit operations would have consumed at most 2^{50} joules. Spreading 2^{50} joules over a year means 2^5 megawatts. At 2^6 watts per chip, 2^5 megawatts means half a million chips, equivalent to a few hundred million dollars at Intel's original $775 sales price (see [136]) for the 266 MHz Pentium II.

For comparison, a 1997 news report [85] said that Intel "is shipping close to 100 million chips a year". Intel's annual report in early 1998 [137, pages 3 and 26] said that "sales of Pentium Pro and Pentium II microprocessors became an increasing portion of the Company's revenues and gross margin in 1996 and a significant portion in 1997", and that Intel's net revenues in 1997 were $25 billion. Intel was already manufacturing earlier "CPUs in volume on a 0.35-micron process" in 1995, according to [122]; the 266 MHz Pentium II was also manufactured at 350nm, according to [136]. There is no reason to think that producing half a million 350nm CPUs for an attack would have run into any manufacturing limits.

D.3 NSA resources

The Federation of American Scientists used public data to conclude in 1996 [98] that the "NSA budget is around $3.6 billion", including "roughly 20,000 direct-hire NSA staff". Even if personnel expenses for an average staff member were as high as $100000, NSA would have had $1.6 billion in 1996 to spend on equipment.

Declassification requests by journalists led to partial declassification in 2013 of internal NSA history books from 1998 and 1999. These books confirm the 20,000 number; see, e.g., [145, page 23]. These books also say [146, page 291]

that NSA spent $199 million in 1984 on a single contract to buy 21,000 IBM PC XTs so as to put a PC on each desk; that NSA spent $150 million in 1985 on a single network-hardware contract; and that "computer power was the essential ingredient in cryptanalysis".

Spending a few hundred million dollars in 1996 on chips to carry out attacks would have been enough to carry out 2^{86} bit operations in 1997 (see Appendices D.1 and D.2), even if there were no contributions from chips bought in previous years, and would have been only 10% of NSA's budget. There is no evident reason that NSA would not have spent even more than this on such chips, say 30% of its budget, in which case an attack consuming 2^{86} bit operations would have been only one of multiple large-scale attacks that NSA could have afforded to carry out at the same time.

For comparison, news reports in 2013 such as [**227**] indicated that NSA's yearly budget was around $10 billion, with half spent on "management, facilities, and support". A news report in 2012 regarding just one of NSA's computer centers, the Bluffdale center [**18**], indicated that the center cost $2 billion, that construction had begun in 2011, and that the center "should be up and running in September 2013", so that center by itself accounted for close to 10% of NSA's budget for those years.

E Speed

X25519 has been consistently observed to outperform NIST P-256 ECDH, as noted in Section 8.2; similarly, Ed25519 has been consistently observed to outperform NIST P-256 ECDSA. As quantification, this appendix reports measurements of recent software on a spectrum of different CPU cores.

E.1 Data collection

We picked machines with a spread of 10 CPUs introduced over the past decade. Table E.1.1 provides detailed information about the CPU in each machine.

On each machine, we compiled OpenSSL 3.2.2 (released in June 2024 [**189**]) and used OpenSSL's `speed` utility to measure NIST P-256. This utility reports ECDH time, ECDSA signing time, and ECDSA verification time. Key-generation time is not reported (ECDH time means only shared-secret computation), but key generation and signing have the same primary bottleneck, namely single-scalar fixed-base-point multiplication.

The `speed` utility runs experiments on a single CPU core. The numbers in Appendix E.2 are operation counts per second on one core. It is reasonable to expect that a CPU with, e.g., 4 cores can carry out operations 4 times more quickly. This expectation would be invalid with overclocking mechanisms such as "Turbo Boost", since overclocking is much more effective when only one core is active; we disabled those mechanisms.

For X25519 and Ed25519, we measured the performance of lib25519 [**51**] on the same CPUs. To maximize comparability, we performed these measurements

machine	CPU	cores	MHz	microarchitecture	year
`alder`	Intel Core i3-12100	4	3300	Golden Cove	2022
`cezanne`	AMD Ryzen 5 PRO 5650G	6	3900	Zen 3	2021
`jasper3`	Intel Celeron N5105	4	2000	Tremont	2021
`panther`	Intel Core i7-1165G7	4	2800	Tiger Lake	2020
`rome1`	AMD EPYC 7742	64	2245	Zen 2	2019
`pi4b`	Broadcom BCM2711	4	1500	Cortex-A72	2019
`gemini`	Intel Celeron N4020	2	1100	Goldmont Plus	2019
`pi3aplus`	Broadcom BCM2837B0	4	1400	Cortex-A53	2018
`rumba7`	AMD Ryzen 7 1700	8	3000	Zen	2017
`nucnuc`	Intel Pentium N3700	4	1600	Airmont	2015
`samba`	Intel Xeon E3-1220 v5	4	3000	Skylake	2015

Table E.1.1. CPUs used for the measurements in Table E.2.1. "Year" is the year that the CPU was introduced. The machine named `rome1` has two identical CPUs; "cores" is the number of cores per CPU.

machine	P-256 DH	X25519 DH	P-256 sign	Ed25519 sign	P-256 verify	Ed25519 verify
`alder`	18700	48075	47009	118851	14450	35893
`cezanne`	23184	53152	53564	132900	17918	34131
`jasper3`	6047	9252	15613	23476	4662	7111
`panther`	15526	43282	36826	91401	11855	25868
`rome1`	12920	24068	29073	73521	9809	17326
`pi4b`	3810	10947	8636	13642	2893	3639
`gemini`	2837	4455	7227	11427	2192	3450
`rumba7`	14595	25527	32723	76847	10811	18921
`nucnuc`	2341	3692	5772	9083	1811	2891
`samba`	17698	35981	38669	88640	13170	28468

Table E.2.1. Speed of readily available software for NIST P-256 and Curve25519 on various CPU cores. See Table E.1.1 for descriptions of the CPUs. Each number is for operations carried out per second on a single CPU core, with overclocking disabled.

with OpenSSL's `speed` utility, using a "provider" that plugs lib25519 into OpenSSL.

E.2 Results

Table E.2.1 reports the speed of each operation on each machine. For previous reports of Curve25519 speeds in various environments, often with in-depth analyses of the speeds, see [**23**], [**115**], [**84**], [**32**], [**52**], [**160**], [**167**], [**208**], [**80**], [**92**], [**133**], [**106**], [**171**], [**56**], [**180**], [**181**], [**182**], [**230**], and [**113**].